Ashutosh Sawale

INDEX

Build Tool :- ant, maven
Logging tool :- Log4J
Testing Tool :- Junit
Reporting :- Jasper reports.

Ant Tool

Contents:

⇒ Keeping appl^n/project ready for execution is called build process of the project. In order to automate the complicated, repititive operations of appl^n/project build process we need build tools like Ant, Maven.

⇒ we can also configure Ant software by just having weblogic installation, for this we need to place the weblogic software supplied ant.bat file location in path environment variable

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

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) Down ant 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 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;.

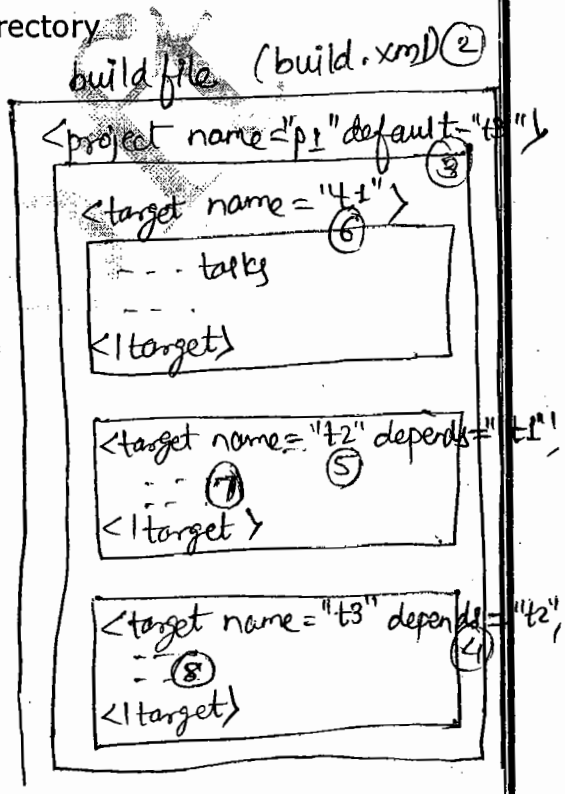
To make sure the installation is proper, go to command prompt and execute the command **ant.**

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

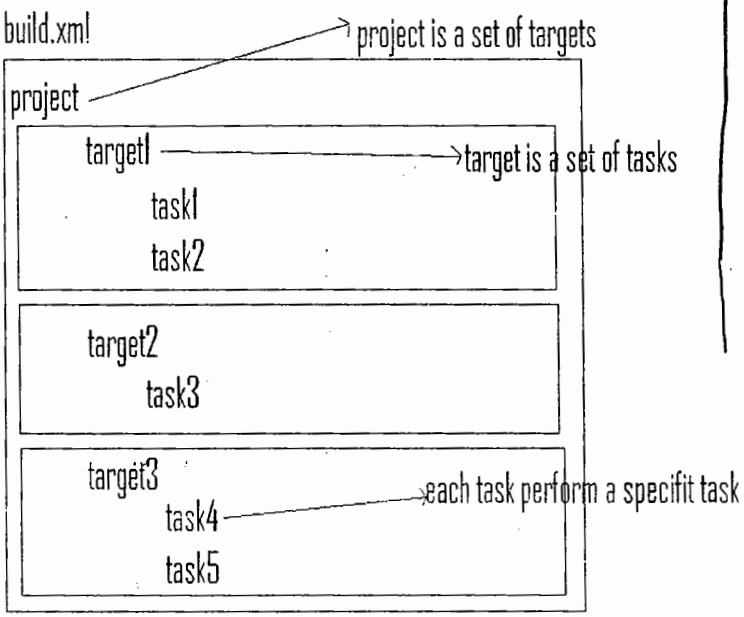
build.xml File

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.
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 use 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

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:

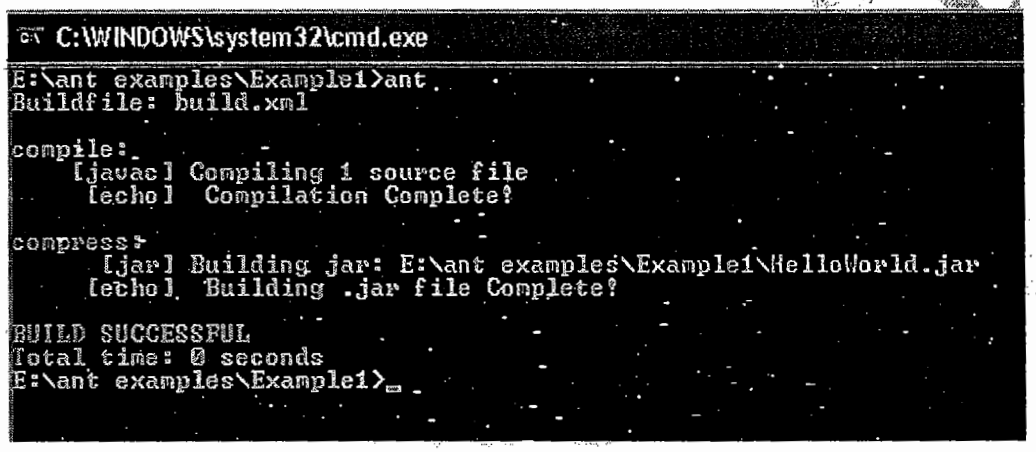
In this example you will see how to compile a java program and compress it into a **.jar** file using Ant build file. The following listing shows the **build.xml** file.

```
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.



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.

```
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>
```

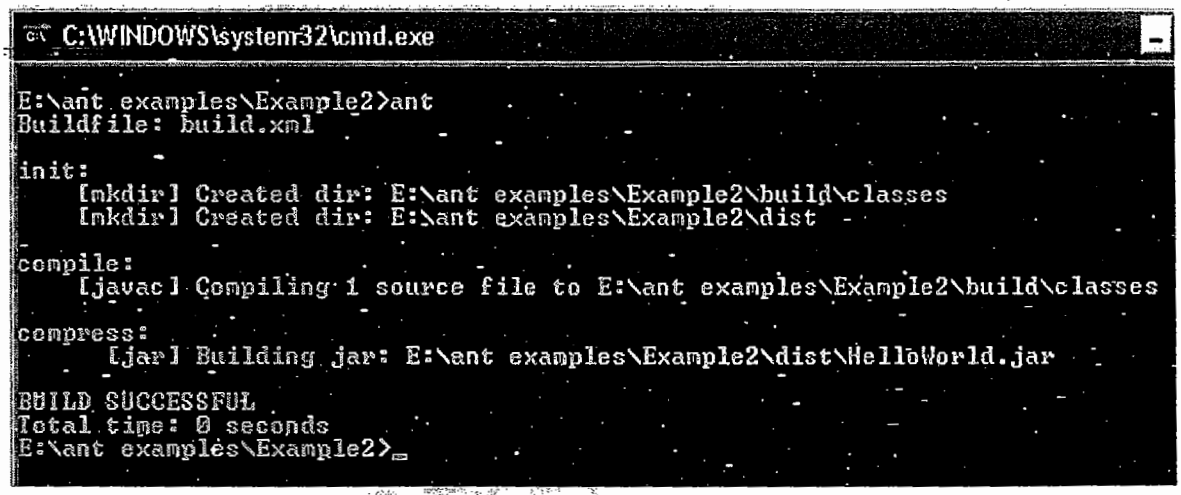
You can use the *java* task to execute a class file as shown below. The **classname** attribute refers to **the java class to be executed** and the **classpath** attribute refers to **the directory in which the class is located.**

```
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.

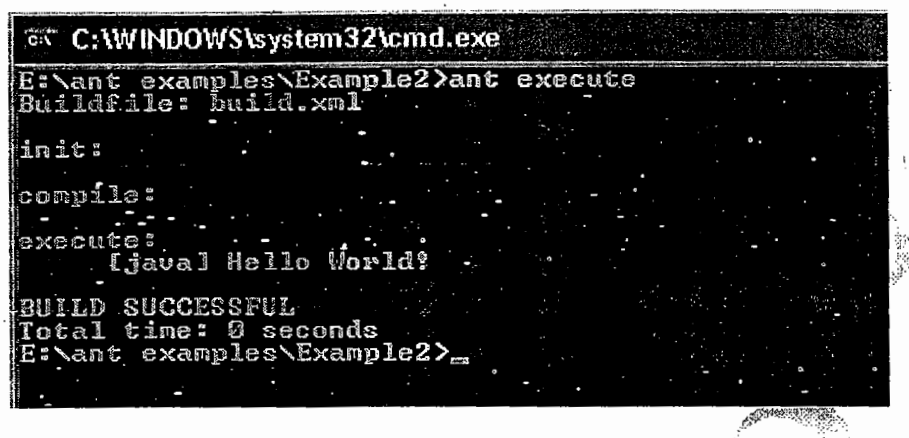
```
1. <target name="clean">
2. <delete dir="build" />
3. <delete dir="dist" />
4. </target>
```

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.

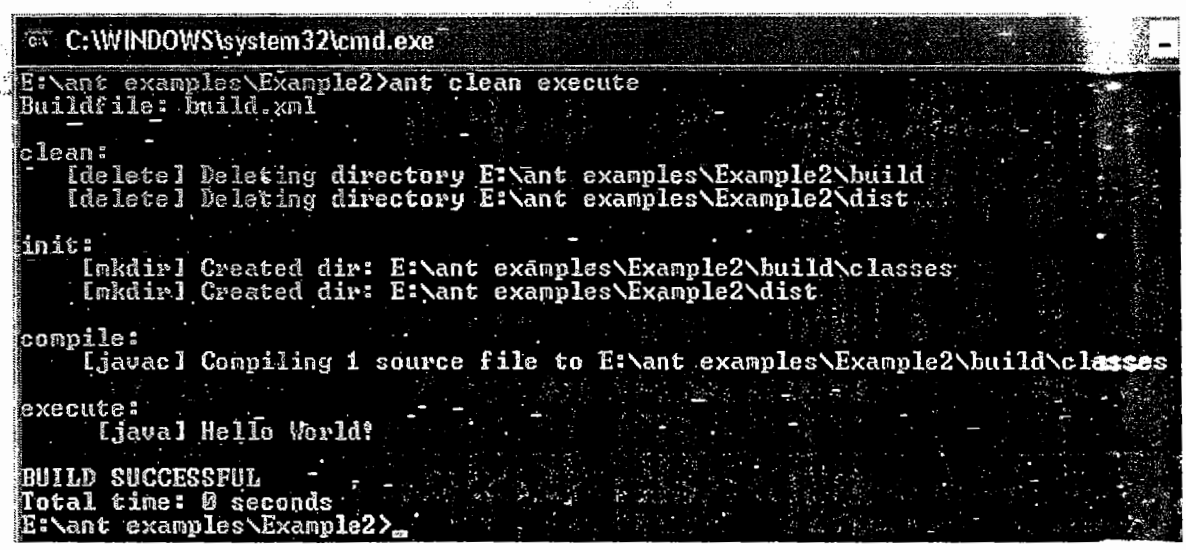


When you run the **"*ant execute*"** command the ***execute*** target will be invoked. ***Execute*** target also depends on ***compile*** target which in turn depends on ***init*** target, but now the directories won't be created again because it already exist. Since the java file is already compiled 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

the console.



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.

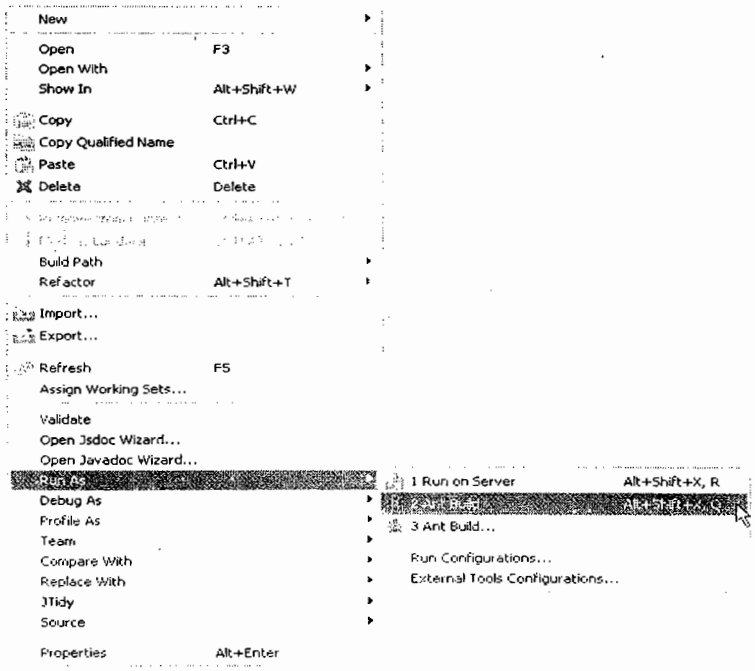


4. Ant Eclipse IDE Integration

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*.

```
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***.



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

```
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. This immutable indicates the value of set in property file cannot be changed through build file if same properties configured with different value after configuring the property file, otherwise this property value is changable.

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?

We use ant taks **<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**

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**

For more build scripts refer pageno:- 272

All the best.....

Using ant tool, you can even perform database operations to prepare dummy tables, dummy data ie. required for project execution/testing. Generally this work will be done by DB team who is not aware of any persistence technologies.

E:\Ant
→ App4
→ file1.sql
→ build.xml

file1.sql

```
create table student10(sno number(5),
  sname varchar2(20), sadd varchar2(20));
insert into student10 values(908,'raja','hyd');
select * from student10
```

E:\Ant\App4>ant ↵

build.xml

```
<project name="test" default="t1"
        basedir="." >
  <target name="t1" >
    <sql driver="oracle.jdbc.driver.OracleDriver"
         url="jdbc:oracle:thin@localhost:
              1521:xe" userid="system"
         password="manager" src="file1.sql"
         print="yes" output="output.txt" >
      <classpath>
        <pathelement location="C:\oracleexe\app\
          oracle\product\11.2.0\server\jdbc\lib
                    \ojdbc6.jar" />
      </classpath>
    </sql>
  </target>
</project>
```

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
  - BasicConfiguror, PropertyConfigurator
  - Setting Logging Level
- ✓ Log4J Logging Levels
  - Default Logging Level
  - Available Logging Levels

    DEBUG, INFO, WARN, ERROR and FATAL
  - Log4J Levels Hierarchy
- ✓ Log4J Logger

- 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

- 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.

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

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 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

- 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.
- log4j is 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

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

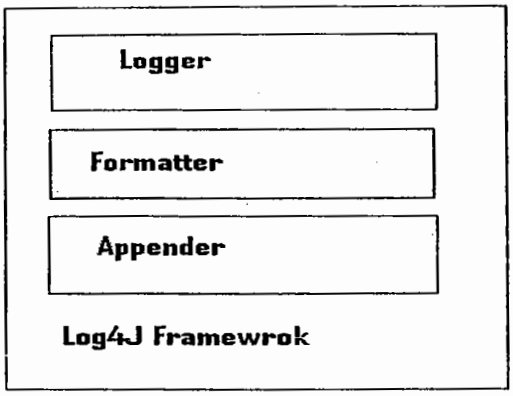
Log4J Logging levels

Level	Description
FATAL	Severe errors that cause premature termination. Expect

	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
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

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

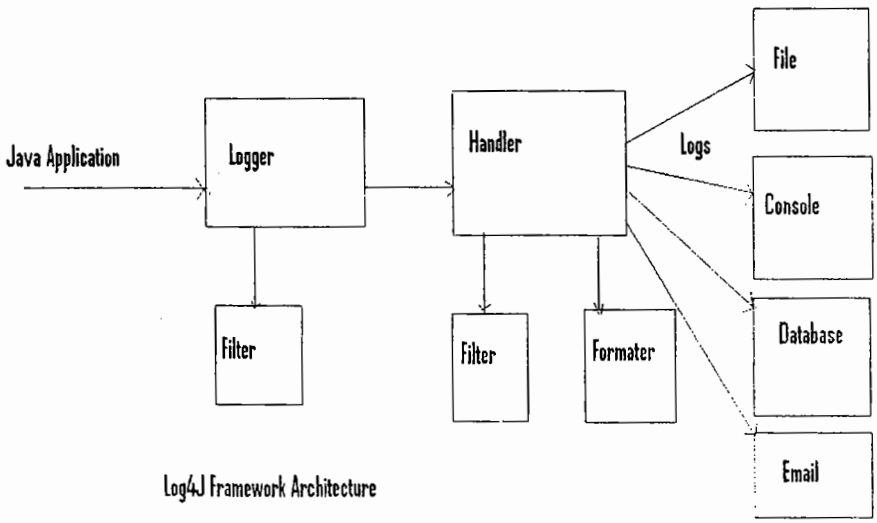


Log4J Frameworks Architecture:

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.



Log4J Framework Architecture

Advantages of Log4j:

– Simpler log level hierarchy and log methods

– 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) User 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.appender.C=org.apache.log4j.ConsoleAppender
log4j.appender.C.layout=org.apache.log4j.PatternLayout

log4j.appender.F=org.apache.log4j.FileAppender
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
```

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
```

In this example, we are using new properties like file size and file backup

log4j.appender.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.appender.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"/>
```

```
    </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

	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.propertiles file?

Yes we can use

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

OR

```
#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
{s
  public static void configure()
  {
    Logger root = Logger.getRootLogger();
```

```
    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 request2,
- ✓ 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
```

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 form 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

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
-%m%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;
```

```
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 -  s%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.

```
[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%n" />
        </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");
}
}
```

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 happing 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 founds 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.

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 whereever 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.

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 Lo4J 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.getRootLogger() API.

LogManager.getRootLogger()

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

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
```

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.

For more example applications on Log4J refer page nos :- 267 to 274

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 approachs?
Advantages of Log4J Declarative approachs?

Why most of the people uses log4j.properties file for logging in ther applicaton?
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.propertiles file?
Is Log4J Thread-safe? Why?
How to solve Performance issues of Log4J?
How and when do loggers inherit level?

SLF4J Example Application:-

step1:- download slf4j-1.7.5.zip file

step2 Develop Application resources

log4j.properties

```
log4j.rootLogger = WARN,F.
log4j.appender.f = org.apache.log4j.FileAppender
log4j.appender.f.File= d:/log.txt
log4j.appender.f.layout = org.apache.log4j.SimpleLayout
```

TestApp.java

```
//import org.apache.log4j.Logger;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
public class TestApp
{
  private static final Logger logger= LoggerFactory.getLogger("TestApp");
  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");
  }
}
```

jar files:- ① log4j<version>.jar ③ slf4j-log4j<version>.jar
② slf4j-api.jar (get from slf4j software)

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 write 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.

Unit Testing comes under white box testing because programmer will do this testing having visibility and accessibility of code.

Example While writing test cases make sure that you are writing one test case for each variety.

- ✓ 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**.
- ✓

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 multiplcation 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

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 ...
```

- ✓ 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 Failure is an anticipated problem that comes when expected results are not matched with actual results.

- ✓ 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. Error is unanticipated problem that comes during business code execution
- ✓ 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

- ✓ 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

- ✓ 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()
  4. clean-up after a test by overriding tearDown().

- ✓ 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 uninitailiztion.
- ✓ Every Test case class must and should extend the TestCase class
- ✓ Every Test case class should follow the following conventions

  Coding Convention :

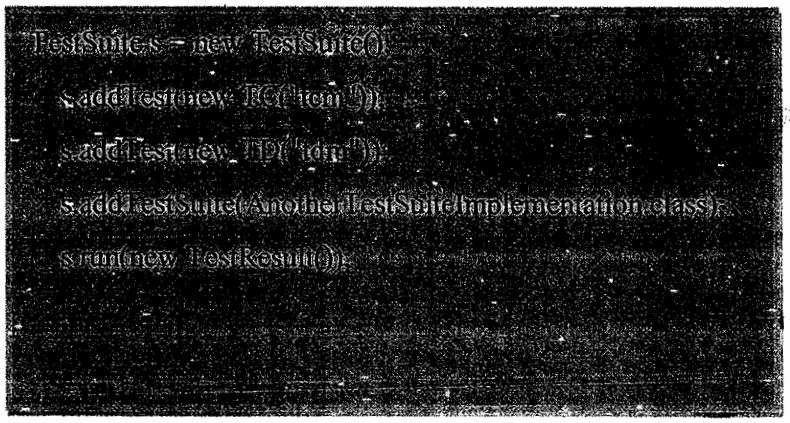
  1. Name of the test class must end with "Test".
  2. Name of the method must begin with "test".
  3. Return type of a test method must be void.
  4. Test method must not throw any exception.
  5. Test method must not have any parameter

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

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.



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.

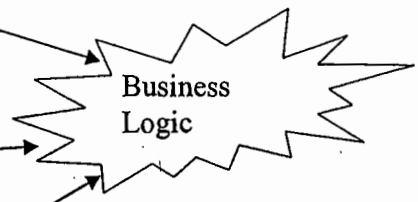
TestRunner prints a trace as the tests are executed followed by a summary at the end.

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

```
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
```



Difference betⁿ Junit 3.x and Junit 4.x.

Junit 3.x	Junit 4.x
1) TestCase class is Api dependent	1) API independent.
2) Does not support Annotations	2) support annotations.
3) No call back methods to execute at the beginning of all tests and at the end of all tests.	3) Available
4) Test ignoring is not possible	4) Possible.

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));
   }

}
```

Initializtion

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>_
```

If TestCae 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".
  - o 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;
```

```
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.

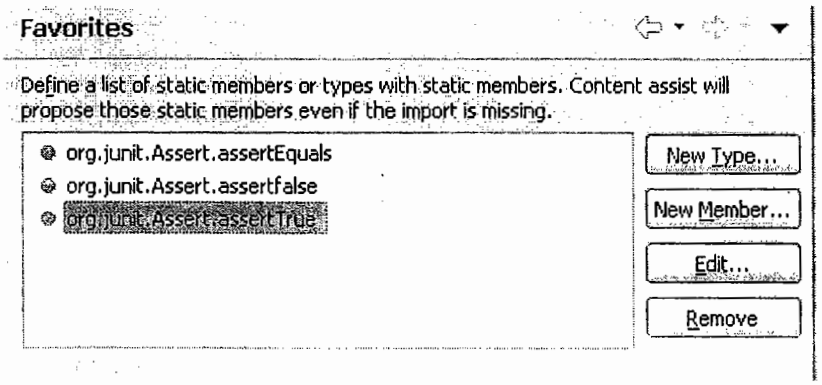
```
package com.dsoft;
import org.junit.runner.JUnitCore;
import org.junit.runner.Result;
import org.junit.runner.notification.Failure;
public class MyTestRunner {
        public static void main(String[] args) {
                Result result = JUnitCore.runClasses(MyClassTest.class);
                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

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
@BeforeClass public void method()	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
@AfterClass 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
@Ignore	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 then 100 milliseconds

Assert statements

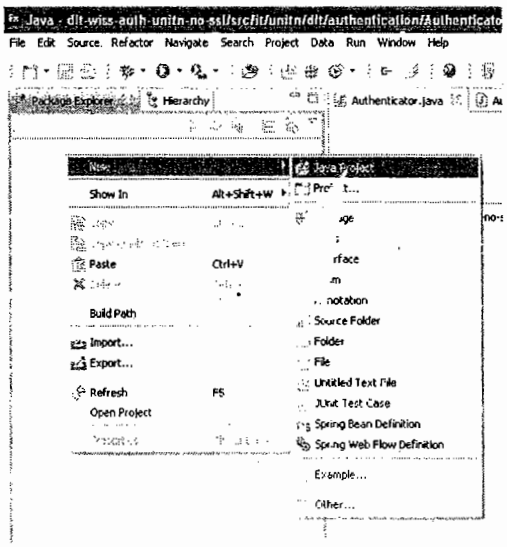
The following gives an overview of the available test methods:

Table 2. Test methods

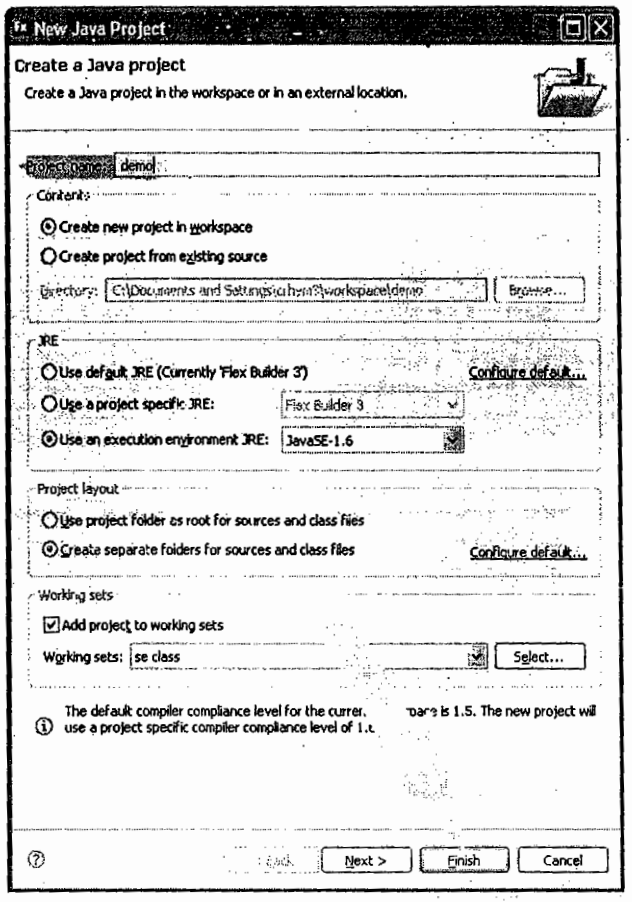
Statement	Description
fail(String)	Let the method fail, might be usable to check

Statement	Description
	that a certain part of the code is not reached.
assertTrue(true);	True
assertsEquals([String message], expected, actual)	Test if the values are the same. Note: for arrays the reference is checked not the content of the arrays
assertsEquals([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.

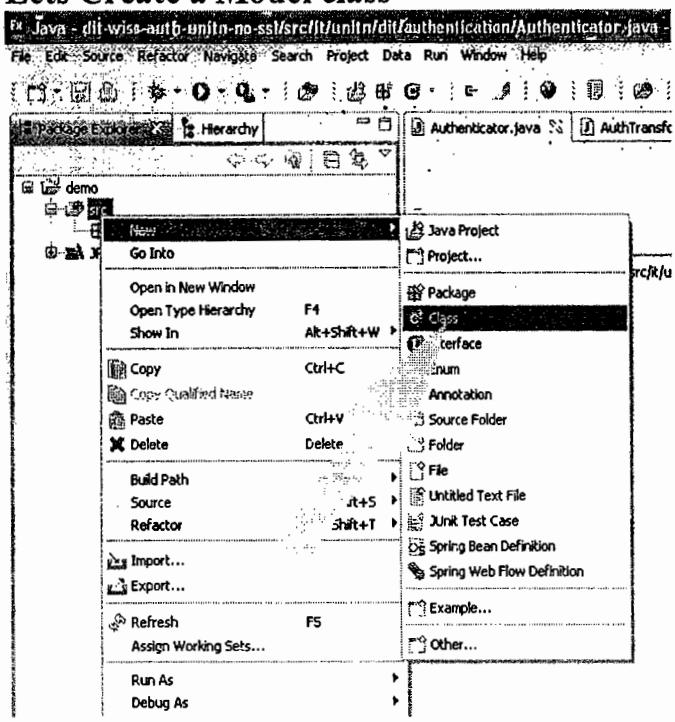
Working with MyEclipse

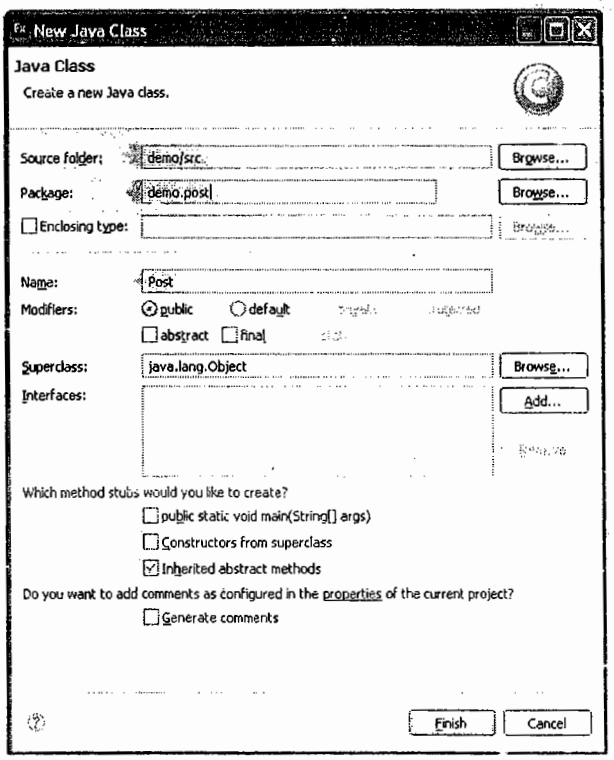


Create a Java Project

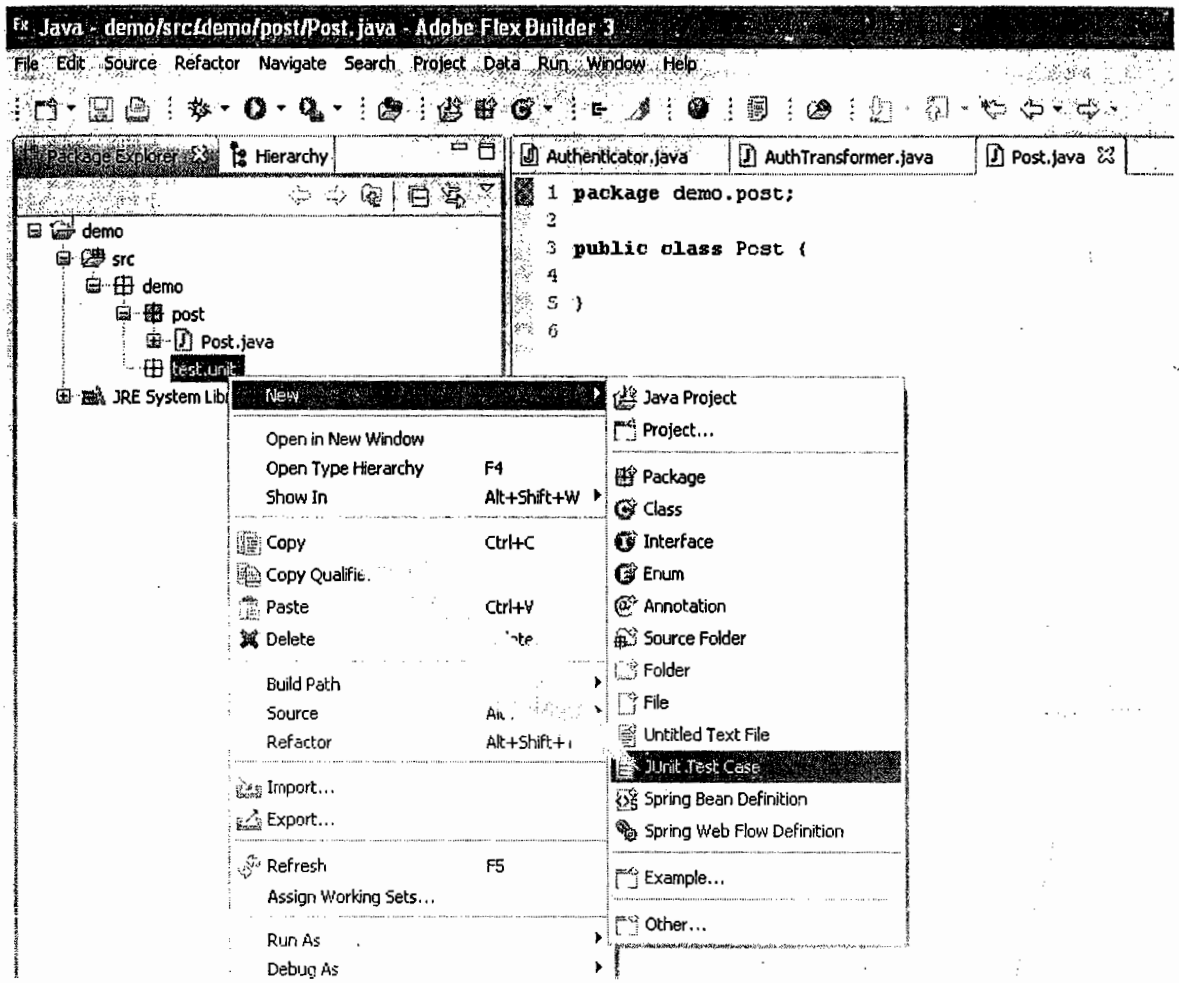

New Java Project
Create a Java project
Create a Java project in the workspace or in an external location.
demo
Contents
Create new project in workspace
Create project from existing source
Browse...
JRE
Use default JRE (Currently 'Flex Builder 3')
Configure default...
Use a project specific JRE:
Flex Builder 3
Use an execution environment JRE:
JavaSE-1.6
Project layout
Use project folder as root for sources and class files
Create separate folders for sources and class files
Configure default...
Working sets
Add project to working sets
Working sets:
Select...
Next >
Finish
Cancel

Lets Create a Model class

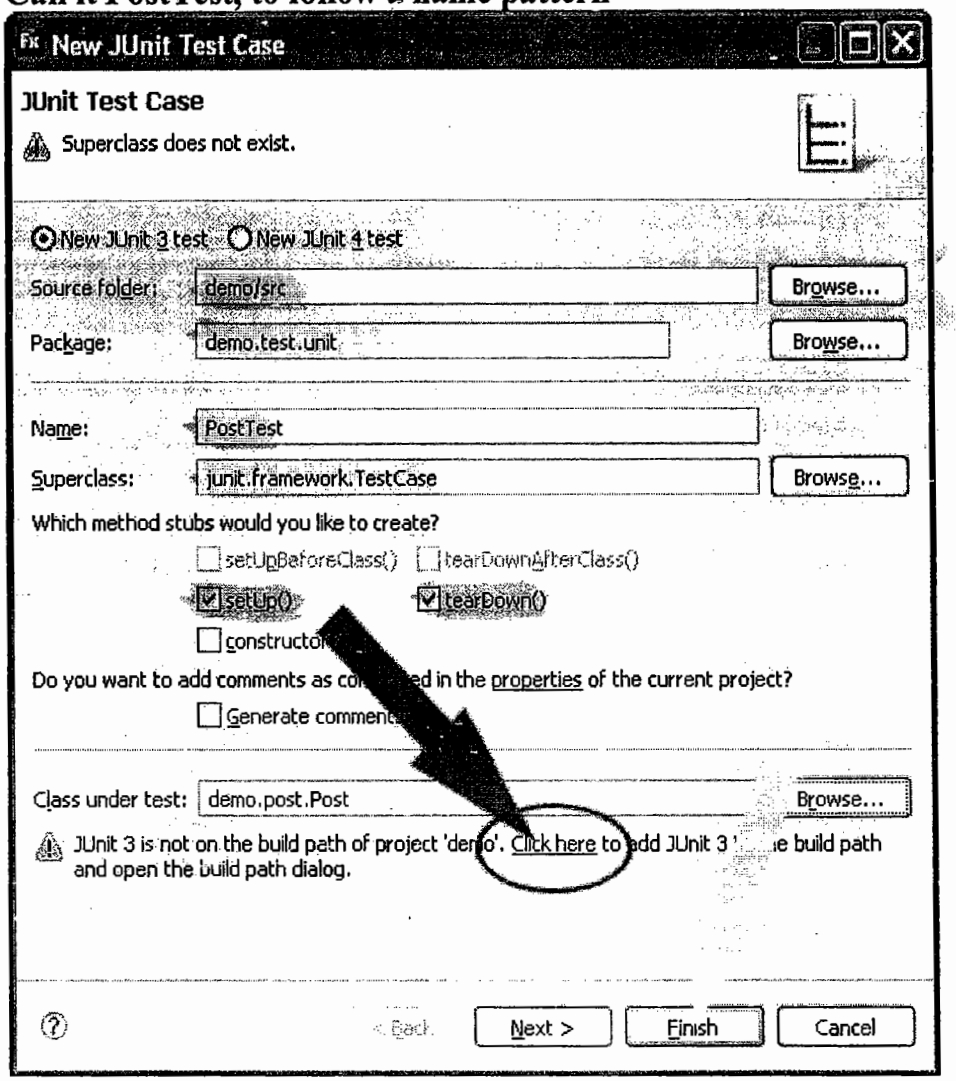




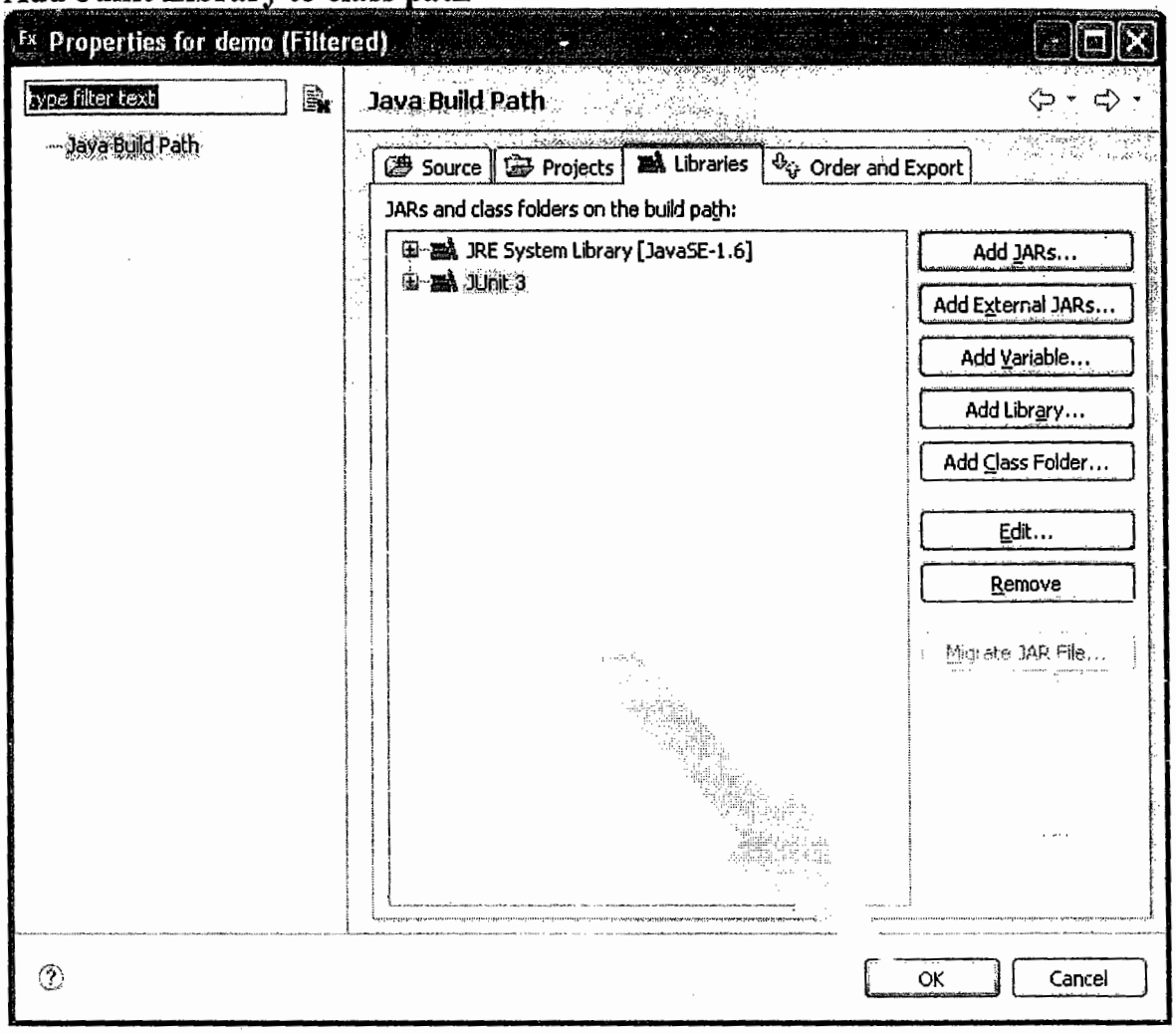
Add a JUNIT TestCase class



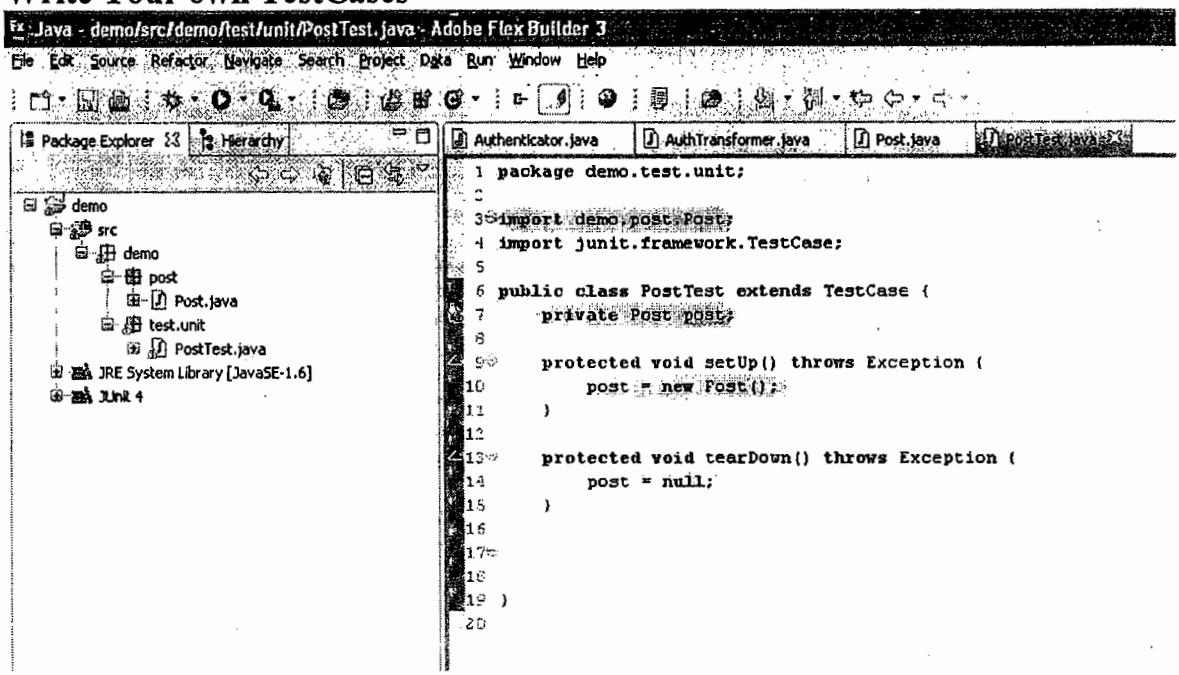
Call it PostTest, to follow a name pattern



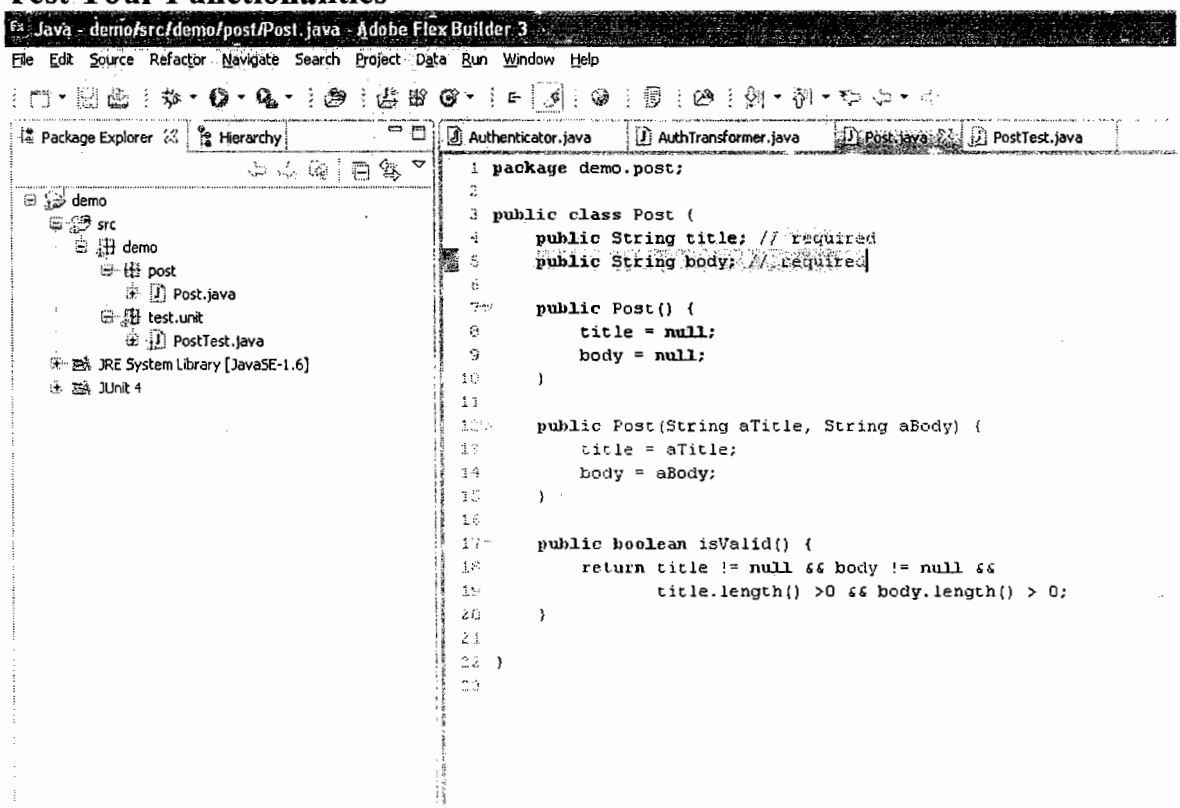
Add Junit Library to class path



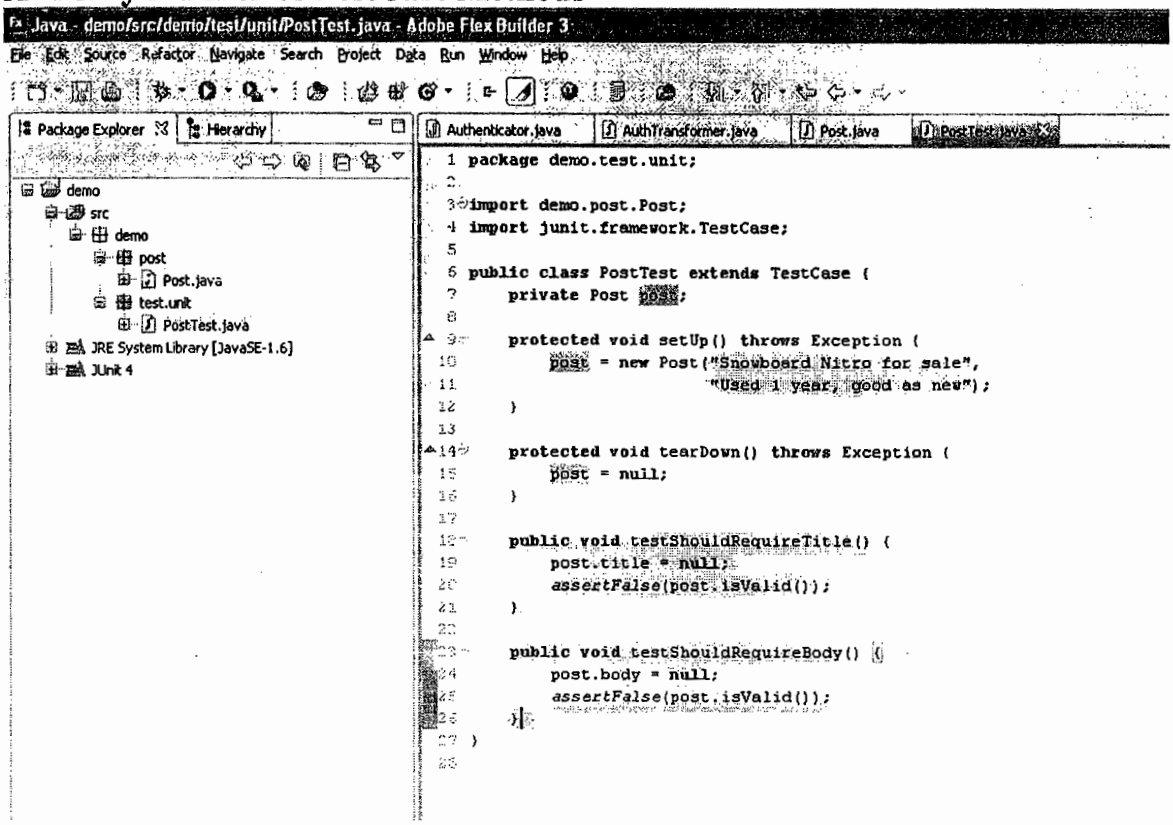
Write Your own TestCases



Test Your Functionalities

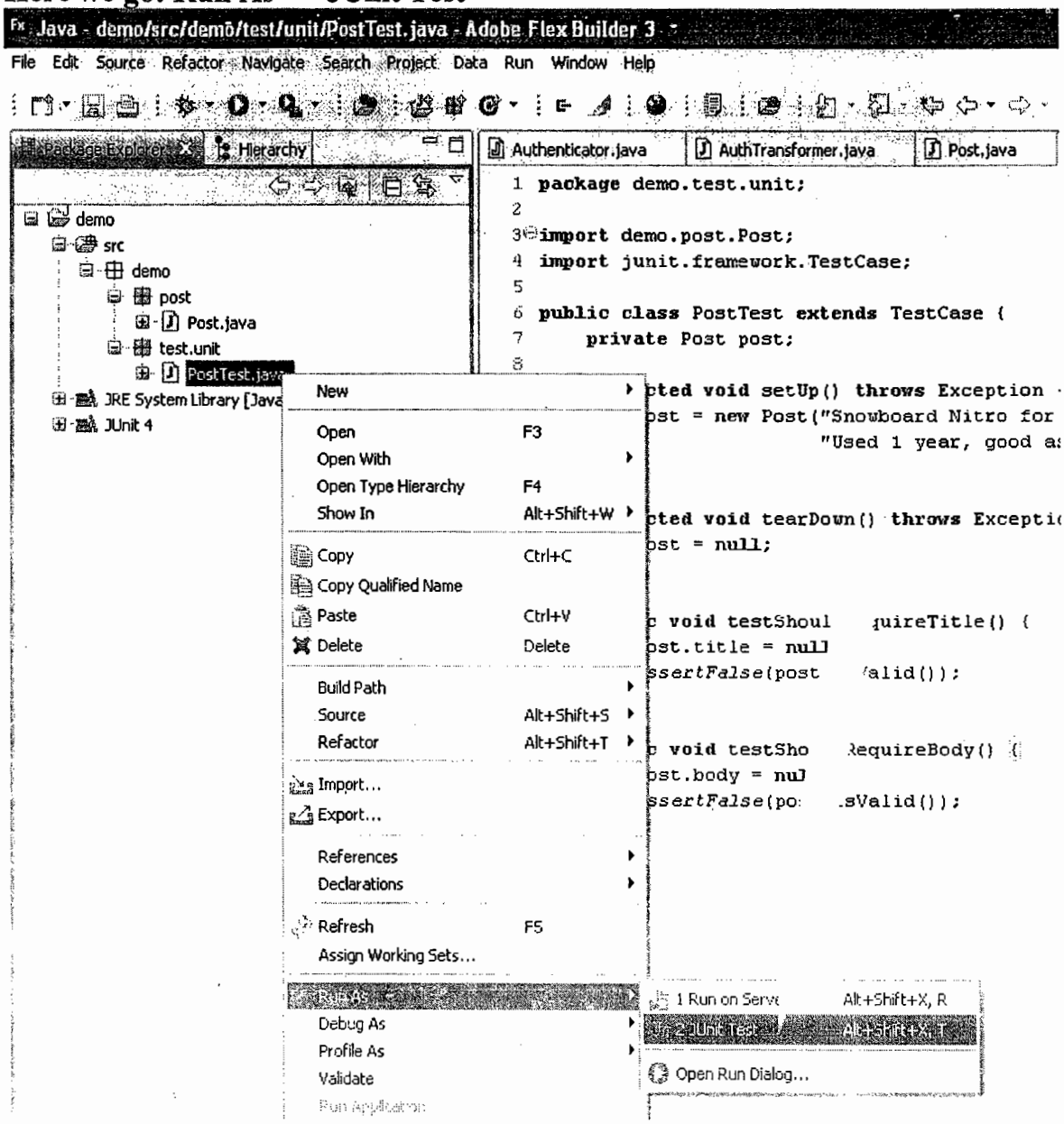


Add Any number of TestCase Methods

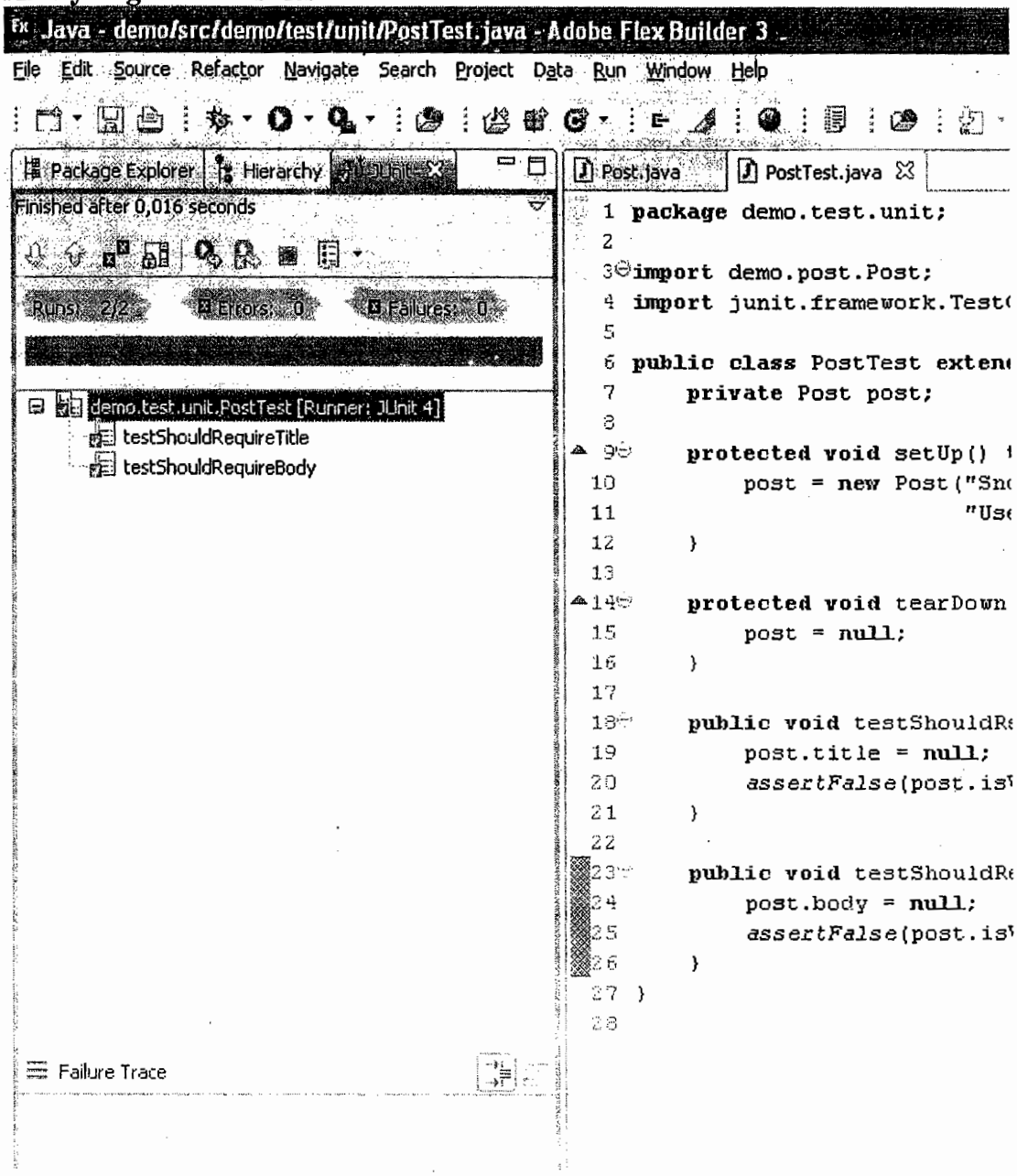


Runnig TestCase

Here we go: Run As => JUnit Test



Analysing Test Results

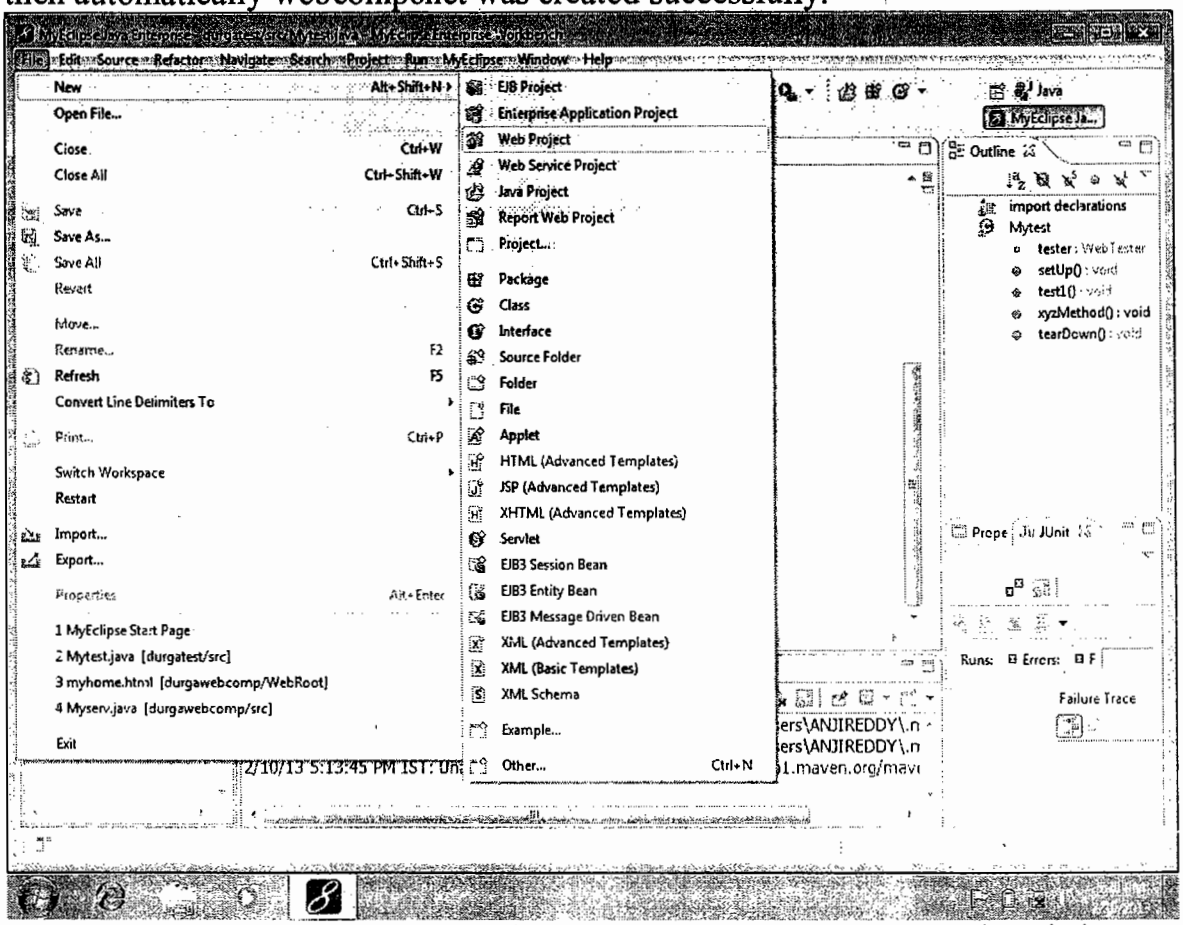


Junit webcomponentdevelopment and Testing Steps

→First of all we need to create new work space(work space is nothing but a small floder)in MyEclipse IDE.

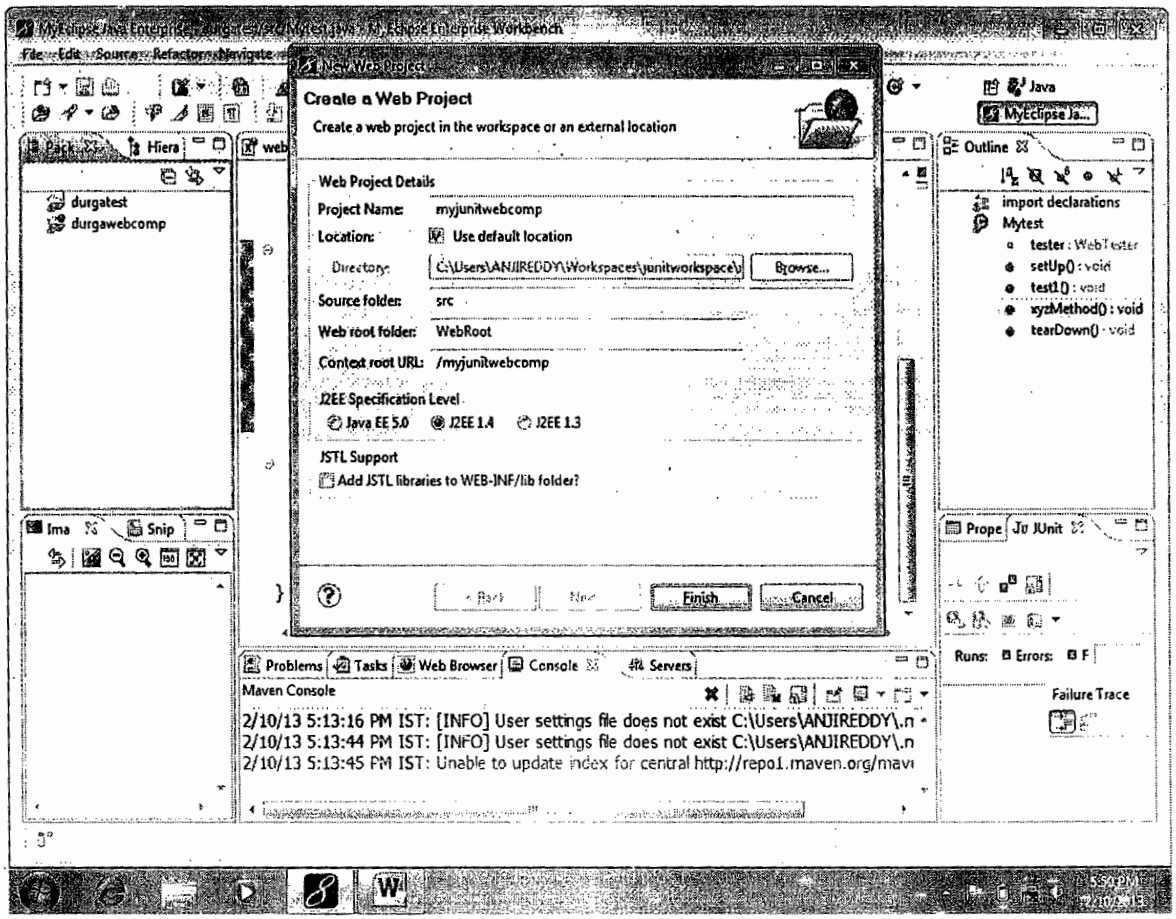
→File->New->webproject->enter the projectname(myjunitwebcomp)->ok->finish.

then automatically webcomponet was created successfully.

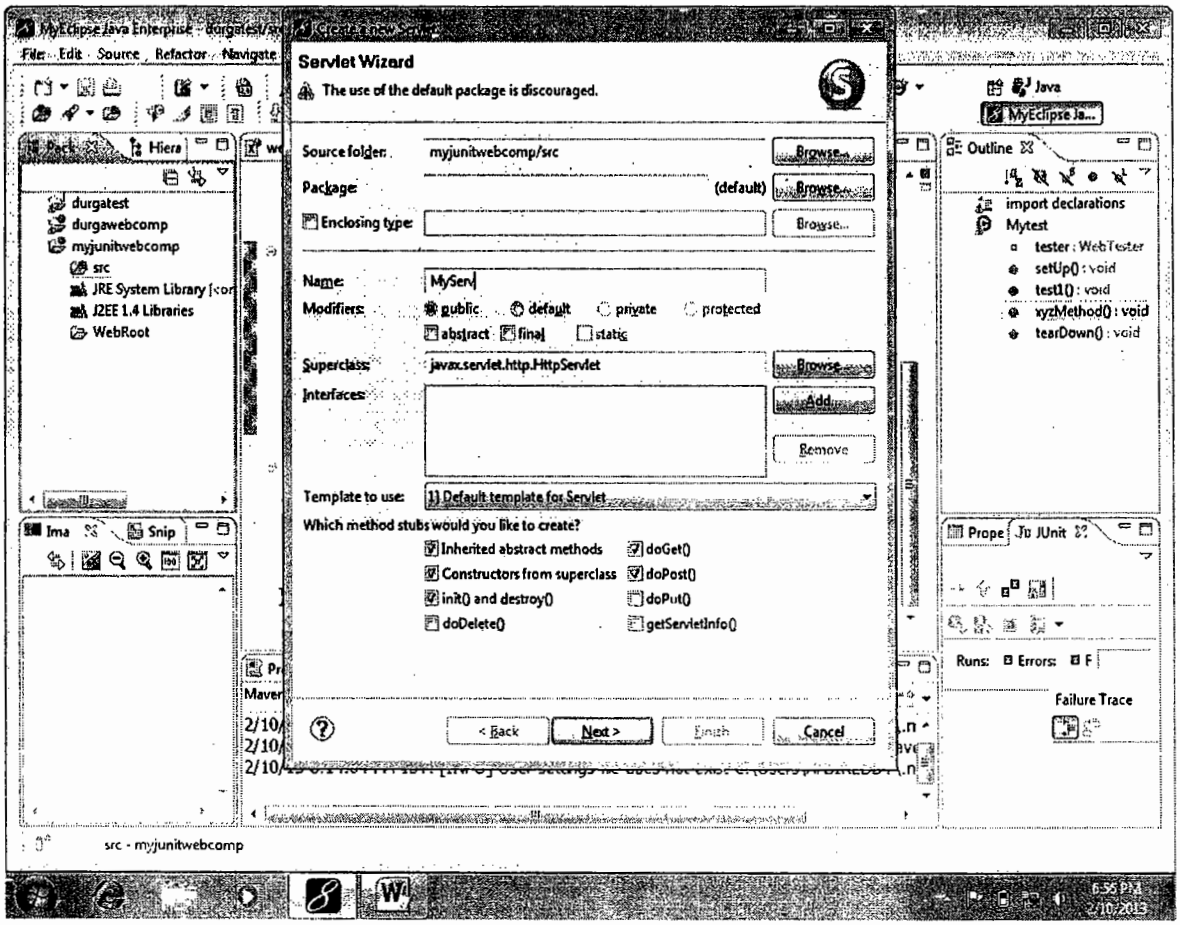


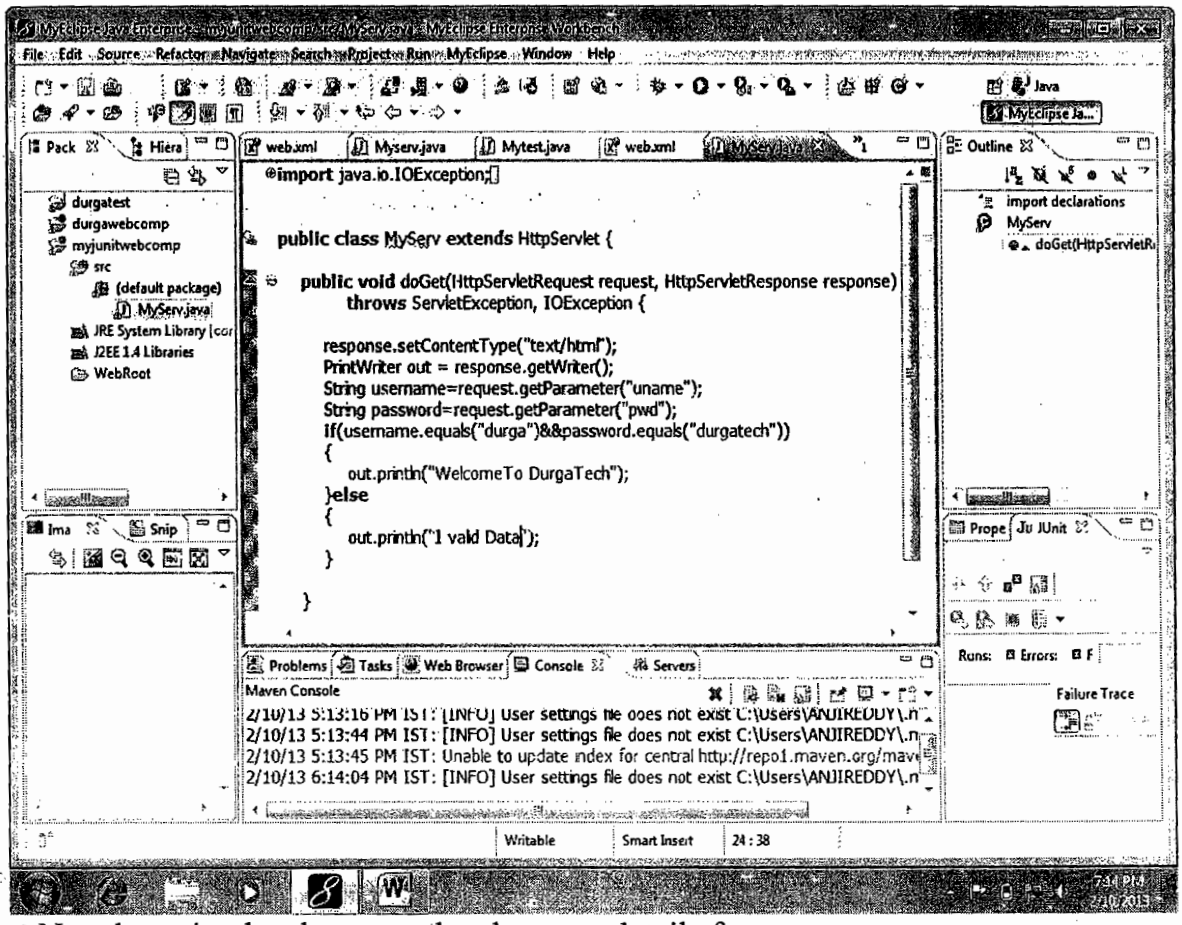
Using Junit you can perform Unit Testing of Normal standalone applications because creating object of service class and calling its methods can be done directly in test case class while writing test. While working with web application to perform Unit Testing we need to send request and we need to get response from each test of TestCase class. To perform these operations without using browser window we need to use special API's like Http Unit, JWebUnit along with JUnit API's.

⇒For example applications refer page no:- 278 and 279

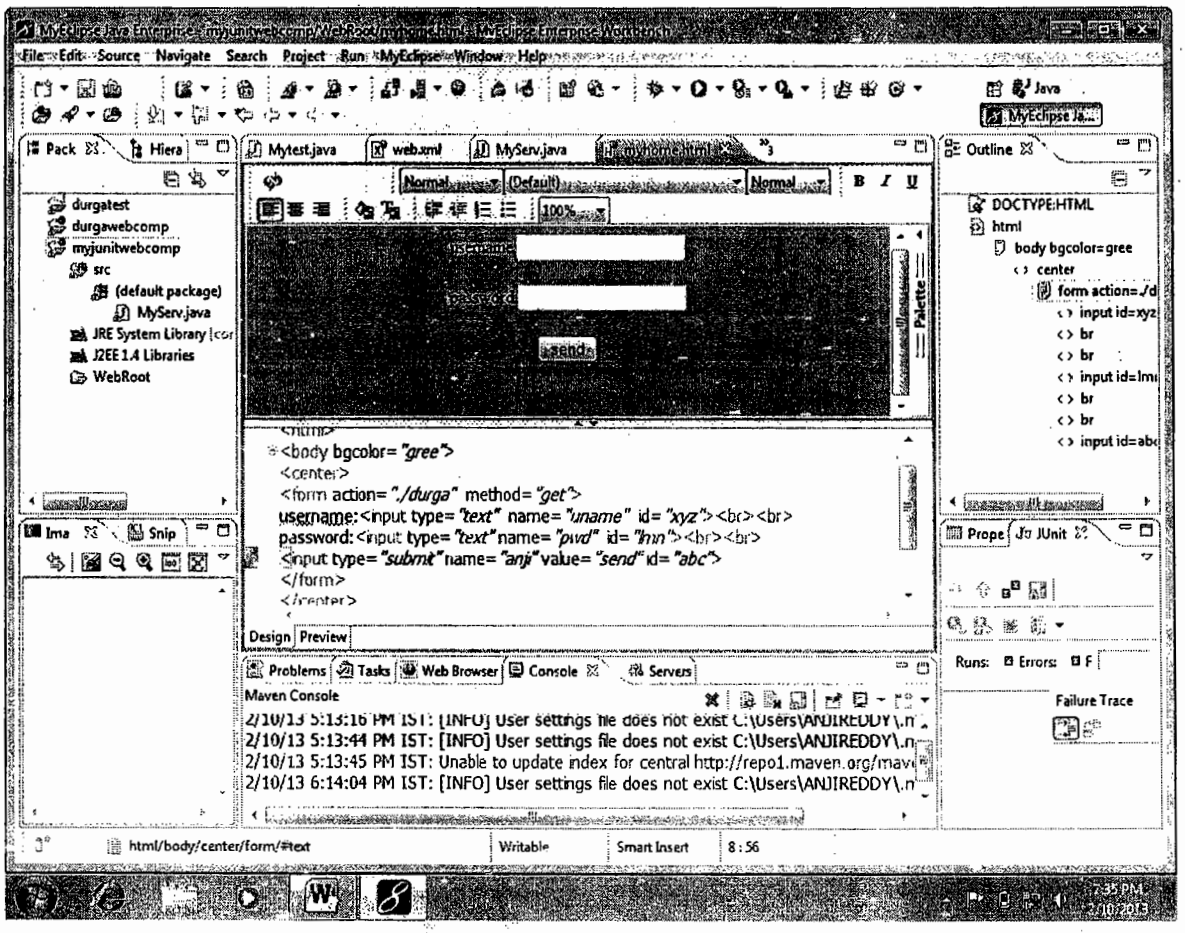


→Now we have to start project development code
→Extactprojectfolder(myjunitwebcomp)->rightclick on src folder->and take one servlet class and we can develop servlet component code.


Servlet Wizard
The use of the default package is discouraged.
Source folder: myjunitwebcomp/src
Package: (default)
Enclosing type:
Name: MyServ
Modifiers: public default private protected
abstract final static
Superclass: javax.servlet.http.HttpServlet
Interfaces:
Add...
Remove
Template to use: 1) Default template for Servlet
Which method stubs would you like to create?
Inherited abstract methods
Constructors from superclass
init() and destroy()
doDelete()
doGet()
doPost()
doPut()
getServletInfo()
< Back
Next >
Finish
Cancel
Outline
import declarations
Mytest
tester : WebTester
setUp() : void
test1() : void
xyzMethod() : void
tearDown() : void
Runs:
Errors:
Failure Trace
src - myjunitwebcomp



→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 manuvally then it is success.
→After that we can go for junit testing and develop the testcases of the webcomponent

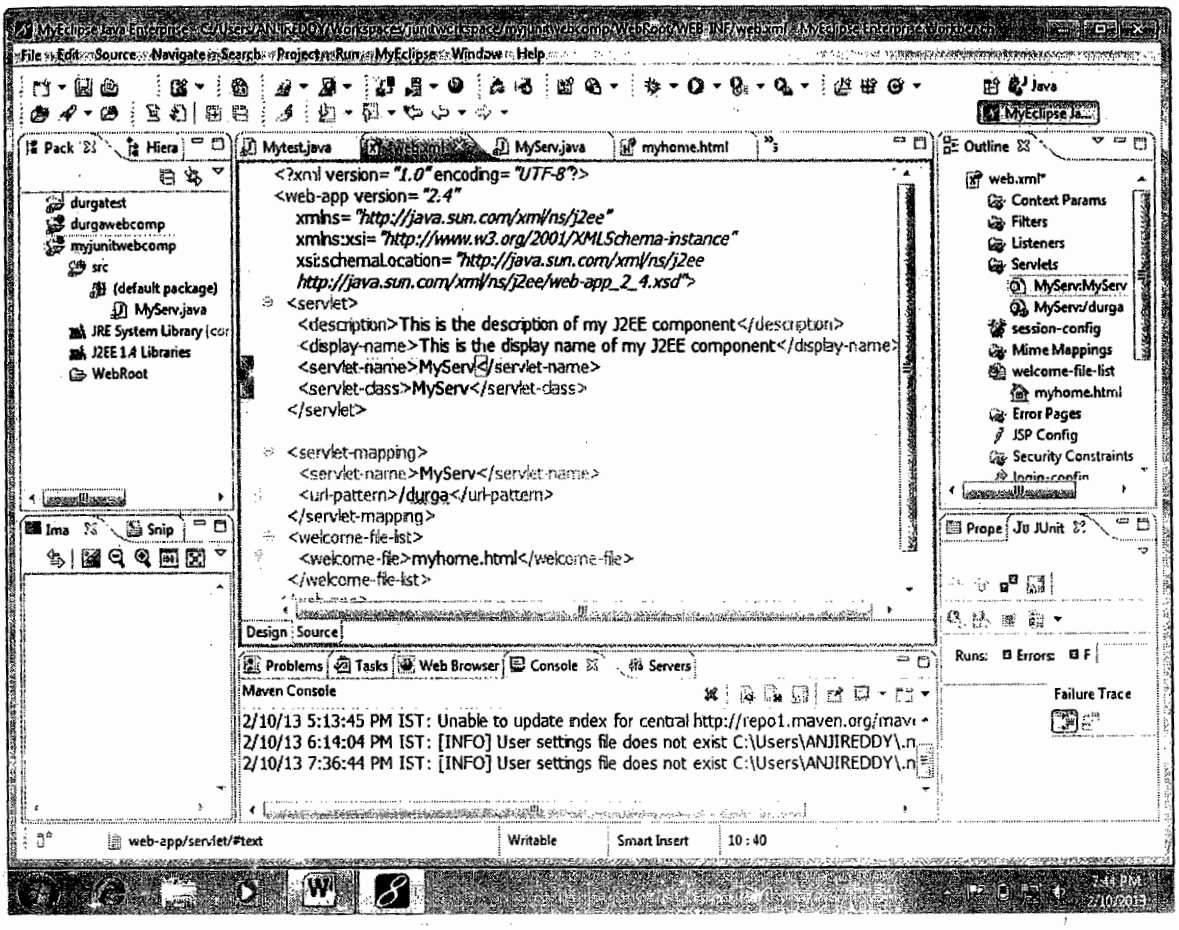

File Edit Source Navigate Search Project Run MyEclipse Window Help
Pack
Hiera
durgatest
durgawebcomp
myjunitwebcomp
src
(default package)
MyServ.java
JRE System Library
J2EE 1.4 Libraries
WebRoot
Mytest.java
web.xml
MyServ.java
Outline
DOCTYPE:HTML
html
body bgcolor=gree
center
form action=./d
input id=xyz
br
br
input id=lm
br
br
input id=abc
<body bgcolor="gree">

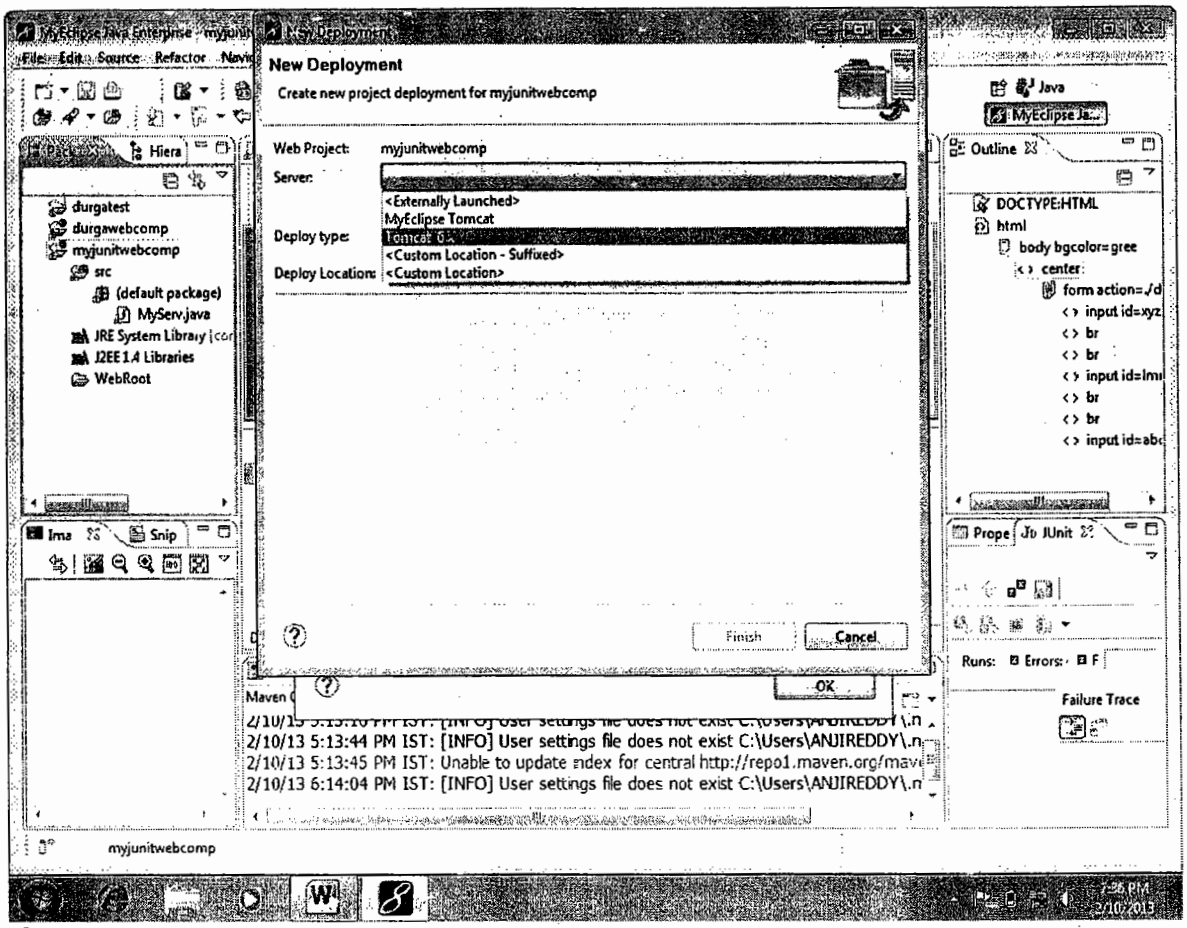
<form action="./durga" method="get">
username:<input type="text" name="uname" id="xyz">

password:<input type="text" name="pwd" id="lmn">

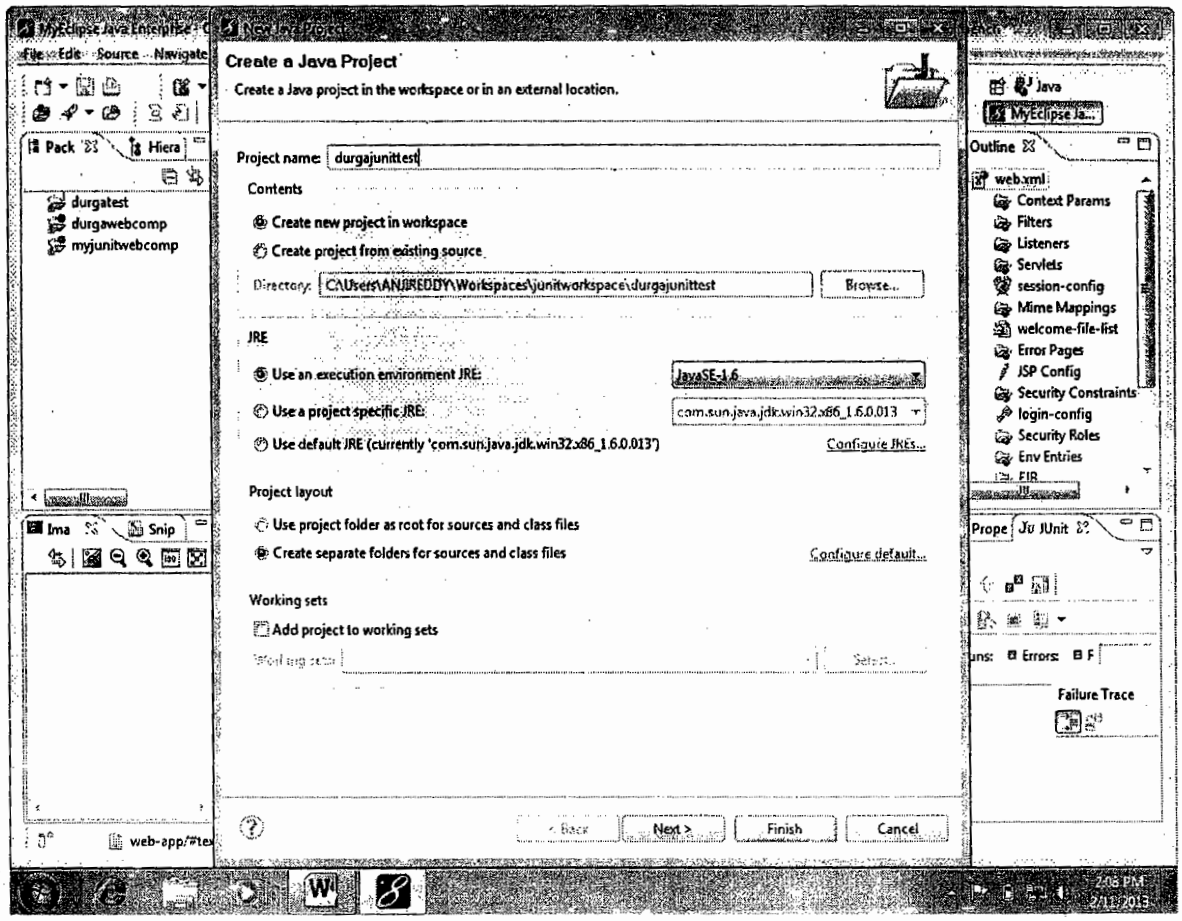
<input type="submit" name="anji" value="send" id="abc">
</form>

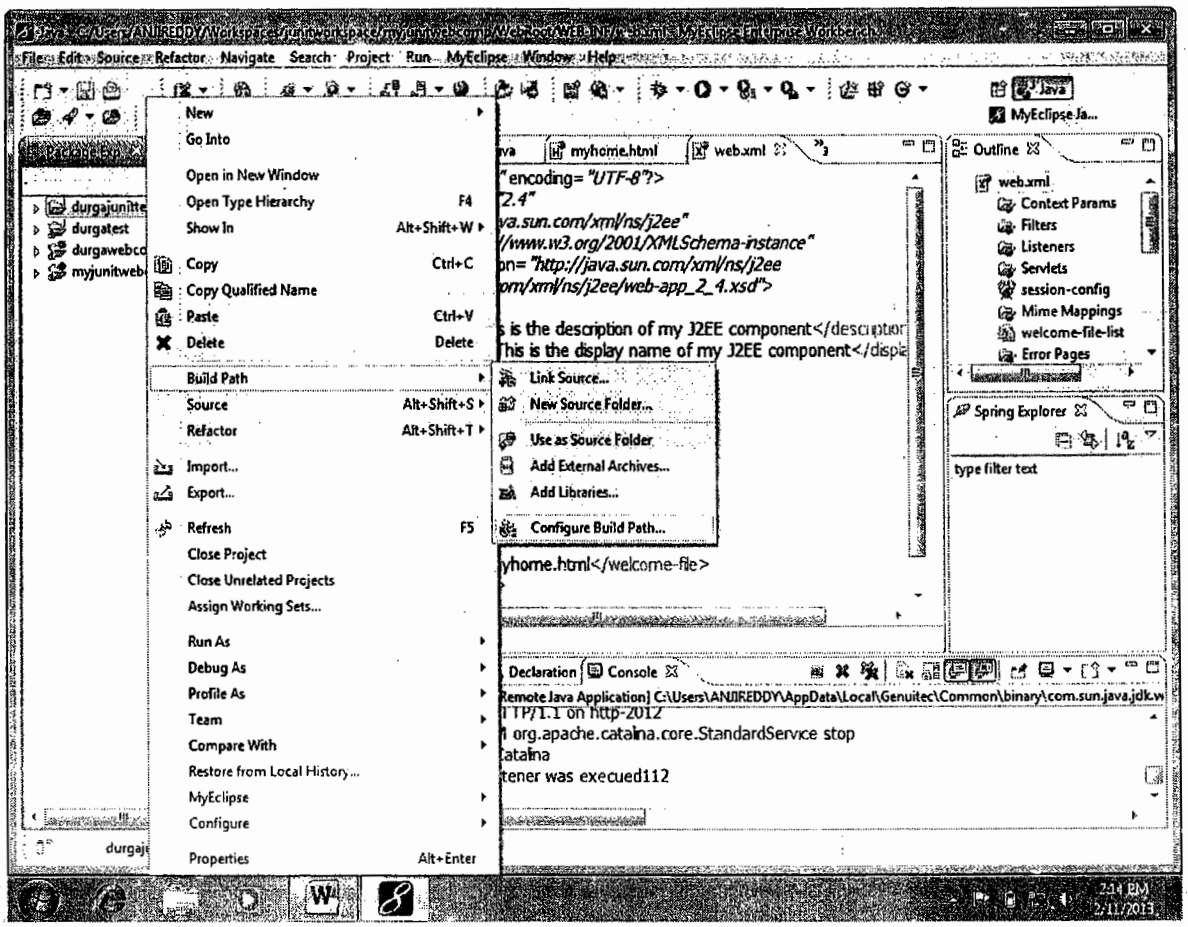
Design
Preview
Problems
Tasks
Web Browser
Console
Servers
Maven Console
2/10/13 5:13:44 PM IST: [INFO] User settings file does not exist C:\Users\ANJIREDDY\.m
2/10/13 5:13:45 PM IST: Unable to update index for central http://repo1.maven.org/mav
2/10/13 6:14:04 PM IST: [INFO] User settings file does not exist C:\Users\ANJIREDDY\.m
Runs:
Errors:
Failure Trace
html/body/center/form/#text
Writable
Smart Insert
8 : 56

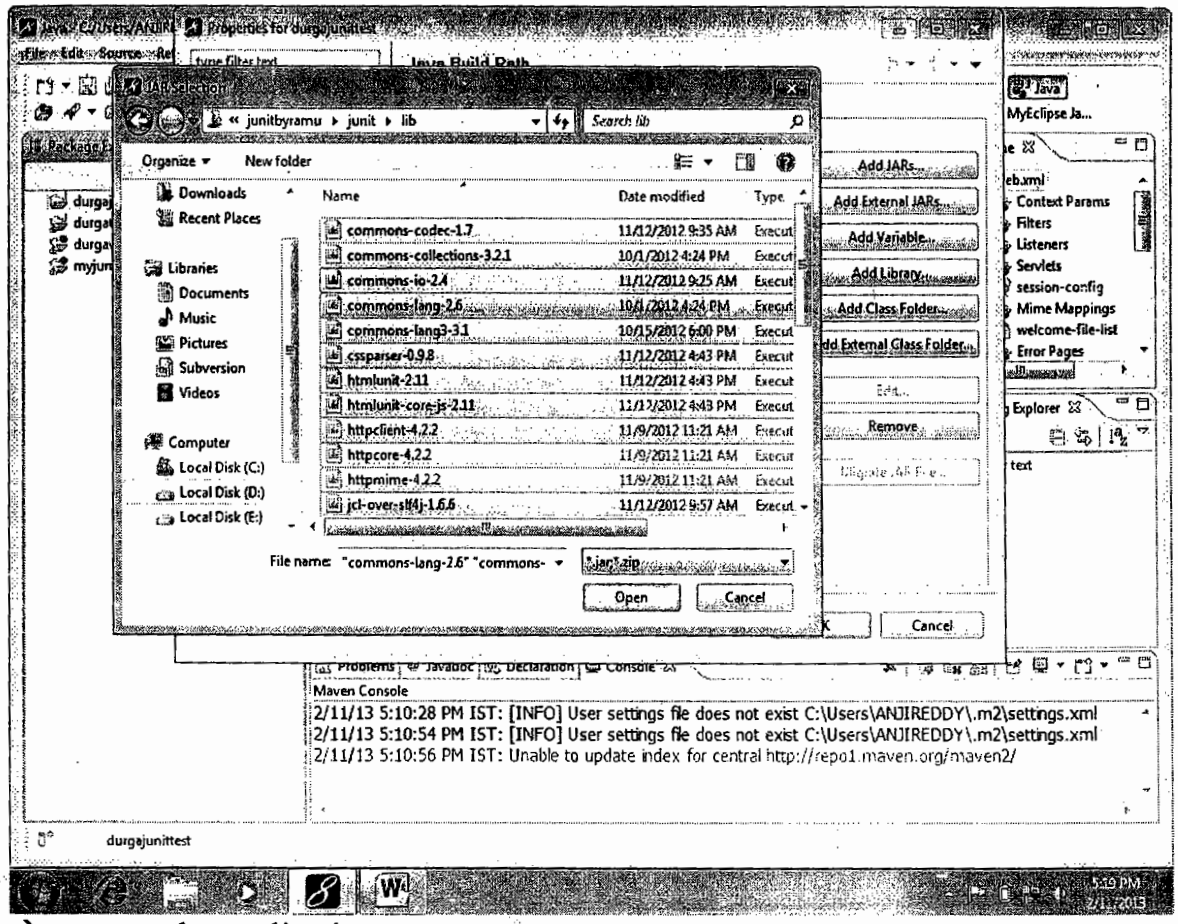

File Edit Source Navigate Search Project Run MyEclipse Window Help
Pack Hiera
durgatest
durgawebcomp
myjunitwebcomp
src
(default package)
MyServ.java
JRE System Library [cor
J2EE 1.4 Libraries
WebRoot
Mytest.java
web.xml
MyServ.java
myhome.html
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4"
xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
<servlet>
<description>This is the description of my J2EE component</description>
<display-name>This is the display name of my J2EE component</display-name>
<servlet-name>MyServ</servlet-name>
<servlet-class>MyServ</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>MyServ</servlet-name>
<url-pattern>/durga</url-pattern>
</servlet-mapping>
<welcome-file-list>
<welcome-file>myhome.html</welcome-file>
</welcome-file-list>
Design Source
Problems Tasks Web Browser Console Servers
Maven Console
2/10/13 5:13:45 PM IST: Unable to update index for central http://repo1.maven.org/mav
2/10/13 6:14:04 PM IST: [INFO] User settings file does not exist C:\Users\ANJIREDDY\.n
2/10/13 7:36:44 PM IST: [INFO] User settings file does not exist C:\Users\ANJIREDDY\.n
Outline
web.xml*
Context Params
Filters
Listeners
Servlets
MyServ:MyServ
MyServ:/durga
session-config
Mime Mappings
welcome-file-list
myhome.html
Error Pages
JSP Config
Security Constraints
Prope JU JUnit
Runs: Errors:
Failure Trace
web-app/servlet/#text
Writable
Smart Insert
10 : 40



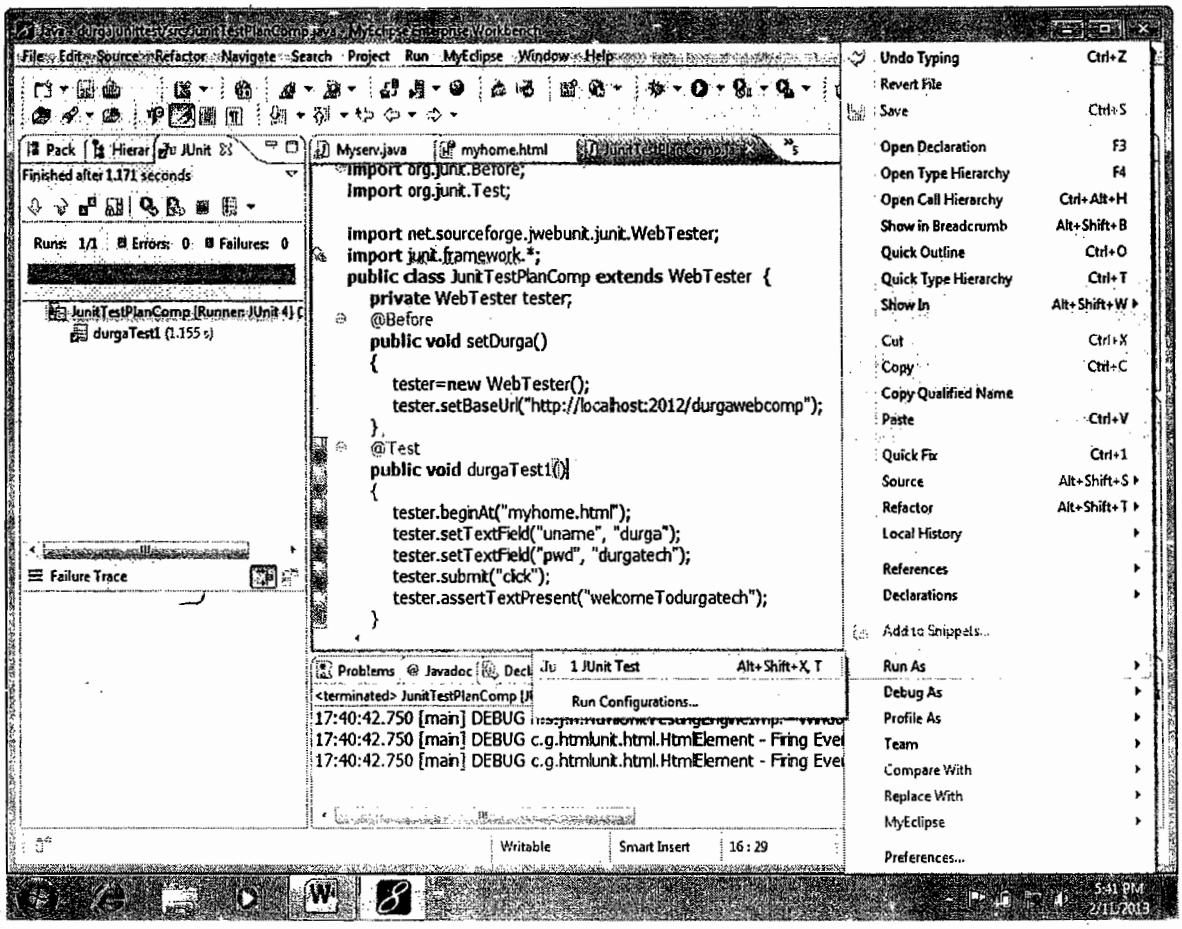
→Now we have to start junit component
→create normal java project and add all the junit related jar files as followes
→file->new->javaproject->enterprojectname(ex:durgajunitproject)->next->finish.
→right click on project->buildpath->cofigurebuildpath->addlibraries ->addexternaljarfiles->add all jar files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

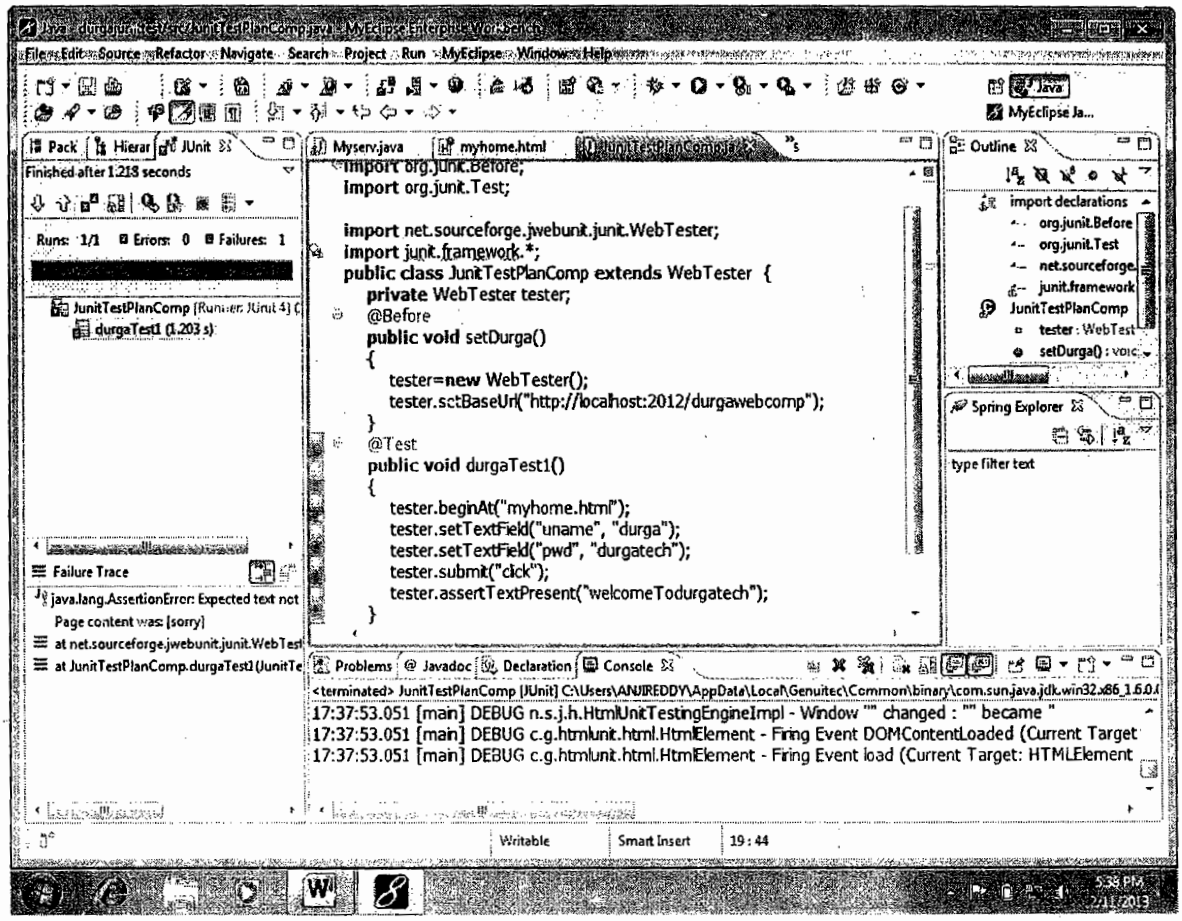

Create a Java Project
Create a Java project in the workspace or in an external location.
Project name: durgajunittest
Contents
Create new project in workspace
Create project from existing source
Directory: C:\Users\ANJIREDDY\Workspaces\junitworkspace\durgajunittest
Browse...
JRE
Use an execution environment JRE:
JavaSE-1.6
Use a project specific JRE:
com.sun.java.jdk.win32.x86_1.6.0.013
Use default JRE (currently 'com.sun.java.jdk.win32.x86_1.6.0.013')
Configure JREs...
Project layout
Use project folder as root for sources and class files
Create separate folders for sources and class files
Configure default...
Working sets
Add project to working sets
Select...
< Back
Next >
Finish
Cancel
durgatest
durgawebcomp
myjunitwebcomp
web.xml
Context Params
Filters
Listeners
Servlets
session-config
Mime Mappings
welcome-file-list
Error Pages
JSP Config
Security Constraints
login-config
Security Roles
Env Entries
Failure Trace


File Edit Source Refactor Navigate Search Project Run MyEclipse Window Help
New
Go Into
Open in New Window
Open Type Hierarchy F4
Show In Alt+Shift+W
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete Delete
Build Path
Source Alt+Shift+S
Refactor Alt+Shift+T
Import...
Export...
Refresh F5
Close Project
Close Unrelated Projects
Assign Working Sets...
Run As
Debug As
Profile As
Team
Compare With
Restore from Local History...
MyEclipse
Configure
Properties Alt+Enter
Link Source...
New Source Folder...
Use as Source Folder
Add External Archives...
Add Libraries...
Configure Build Path...
myhome.html
web.xml
Outline
web-app
Context Params
Filters
Listeners
Servlets
session-config
Mime Mappings
welcome-file-list
Error Pages
Spring Explorer
type filter text
Declaration
Console
org.apache.catalina.core.StandardService stop
tener was execued112

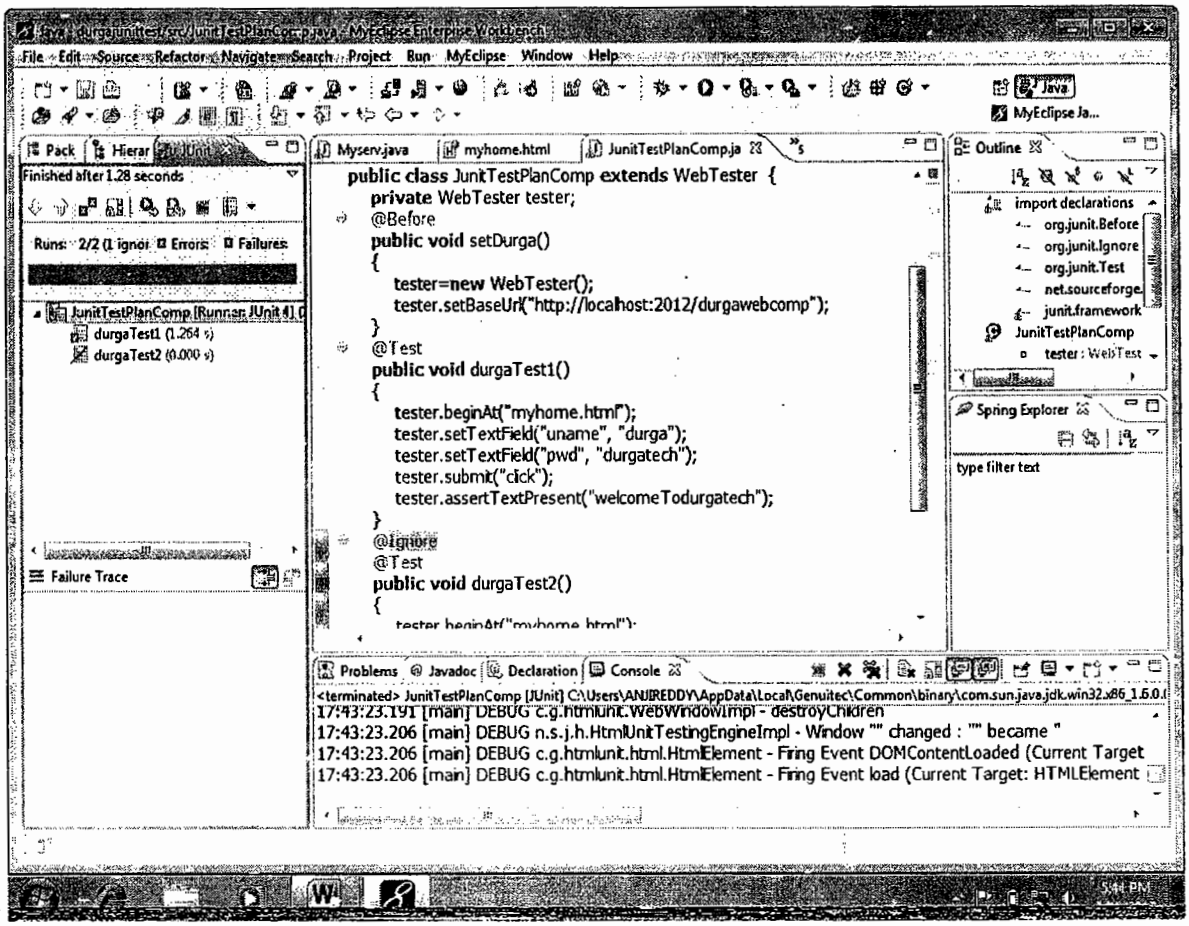


→now test the application.


File Edit Source Refactor Navigate Search Project Run MyEclipse Window Help
Pack Hierar JUnit
Finished after 1.171 seconds
Runs: 1/1 Errors: 0 Failures: 0
JunitTestPlanComp [Runner: JUnit 4]
durgaTest1 (1.155 s)
Failure Trace
Myserv.java myhome.html
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");
}
Problems Javadoc
<terminated> JunitTestPlanComp
17:40:42.750 [main] DEBUG c.g.htmlunit.html.HtmlElement - Firing Eve
17:40:42.750 [main] DEBUG c.g.htmlunit.html.HtmlElement - Firing Eve
Writable Smart Insert 16 : 29
Undo Typing Ctrl+Z
Revert File
Save Ctrl+S
Open Declaration F3
Open Type Hierarchy F4
Open Call Hierarchy Ctrl+Alt+H
Show in Breadcrumb Alt+Shift+B
Quick Outline Ctrl+O
Quick Type Hierarchy Ctrl+T
Show In Alt+Shift+W
Cut Ctrl+X
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Quick Fix Ctrl+1
Source Alt+Shift+S
Refactor Alt+Shift+T
Local History
References
Declarations
Add to Snippets...
Run As
Debug As
Profile As
Team
Compare With
Replace With
MyEclipse
Preferences...
1 JUnit Test Alt+Shift+X, T
Run Configurations...
5:31 PM
2/11/2013



The above case is failure case.



This is an Ignore case

For more example applications on Junit 3.x and 4.x refer page no. 275.

CVS

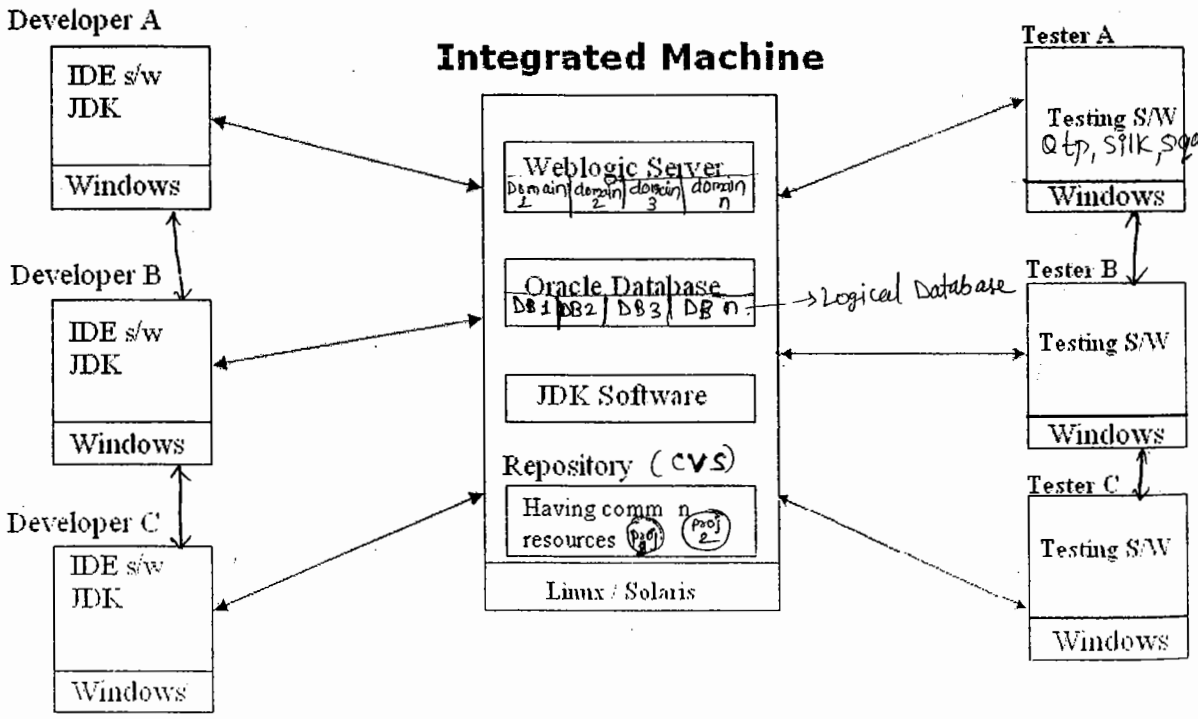
CONTENT:

CVS (Version Control System)

code Versioning 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.

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 : **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: (CVS)

- ✓ All code changes are tracked.
- ✓ Avoid losing work due to simple mistakes (Allows you to roller back changes).
- ✓ Codes 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:

- **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 (checkin)
- **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 → windows compatible
- ✓ WinCVS
- ✓ Tortoise cvs
- ✓ ClearCase cvs , etc. } compatible with Linux, Solaris

❖ 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.

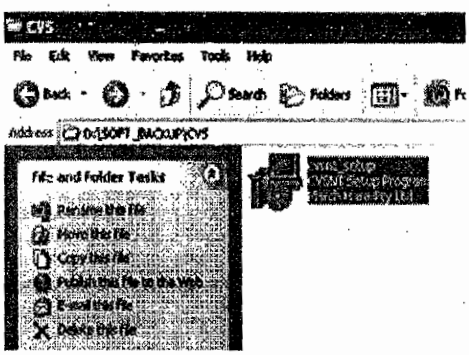
CVS NT :

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

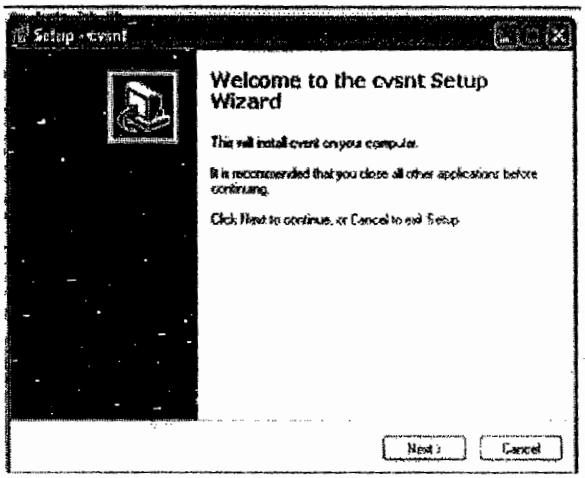
CVSNT INSTALLATION:

SERVER INSTALLATION STEPS

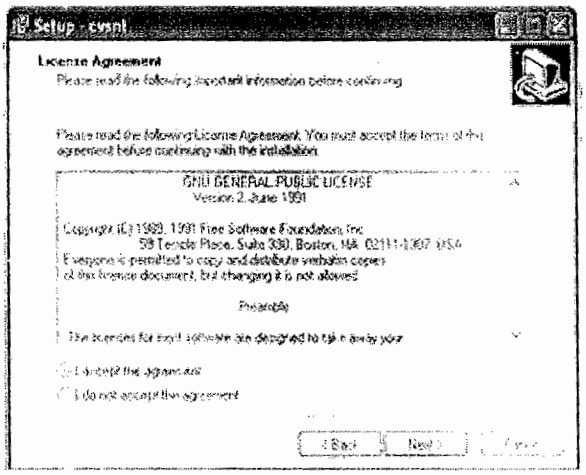
① Installing cvs server softwares like CVSNT on integrated Machine is the responsibility of Administrators. ② Configuring Eclipse IDE of developer Machines with cvs server software and performing checkin, checkout operations is the responsibility of programmers.
③ Project Leader or Team Leader create project having common resources and places that project to the cvs repository and to make it shardable. All programmers checkout that project into Eclipse IDE's of developer Machines.



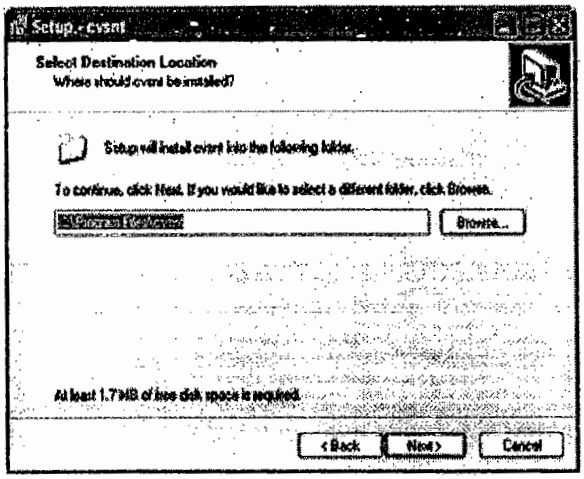
CVSNT setup file.Double Click on the Setup file



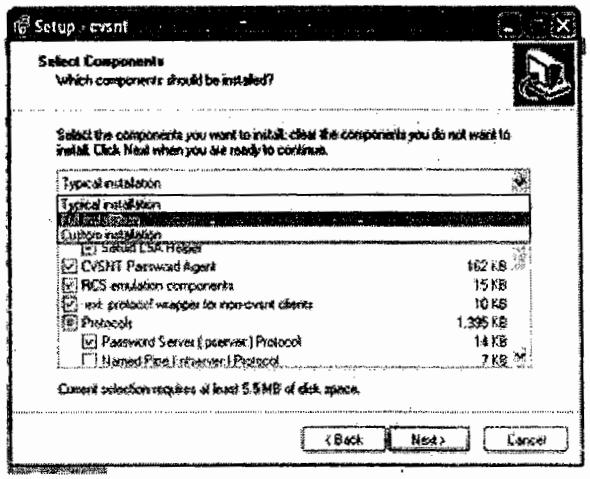
Click on Next



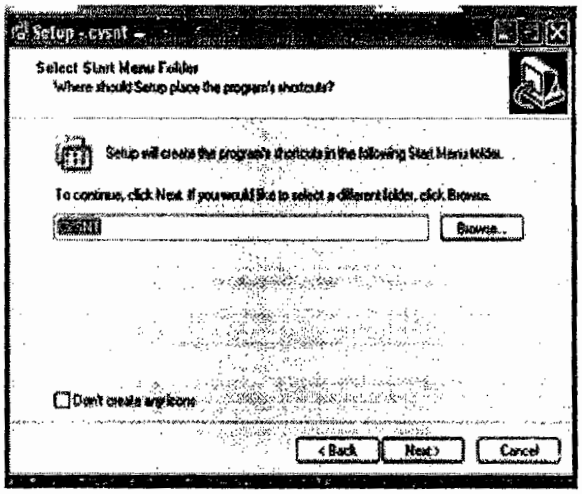
Accept the agreement and click on Next



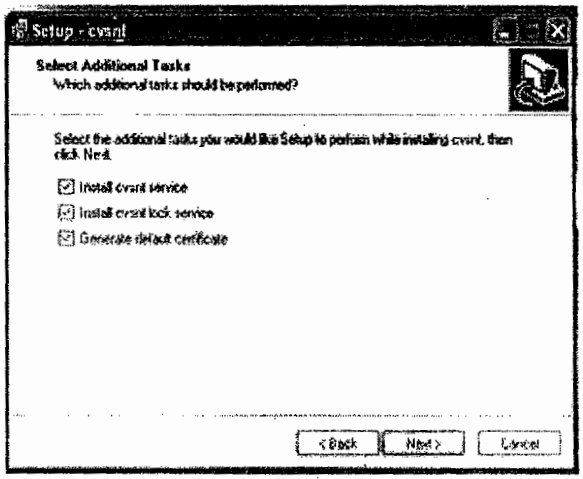
Select the Installation Directory(keep Default Only) and Click on Next



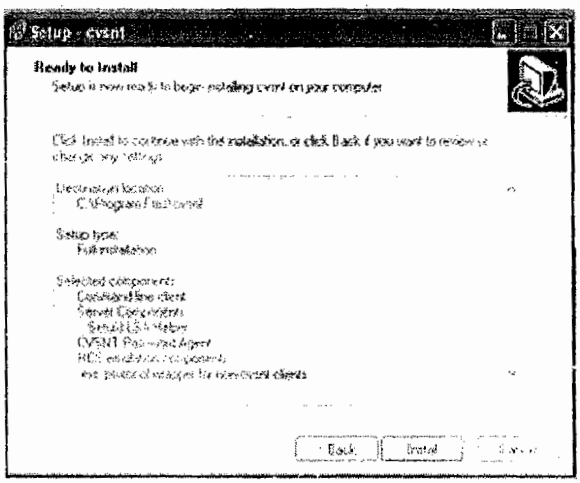
Select Full Installation and Click On Next



Click on Next



Select All and Click on Next



Click on Install

Click on Finish

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

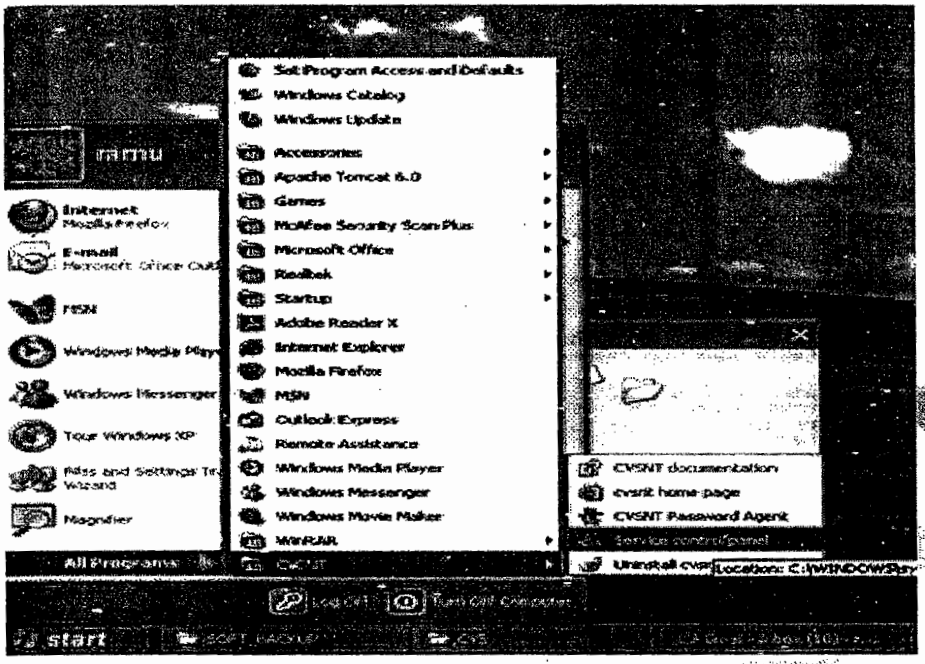
Step-1 : create a directory in computer's file system.

Ex: D:\ CVSRep

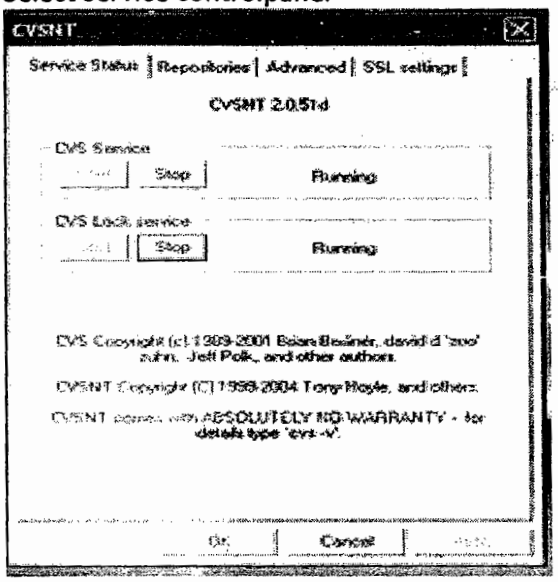
Step-2 : Start the service control panel of CVS NT software pointing to above folder (D:\CVSRep).

" 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 "

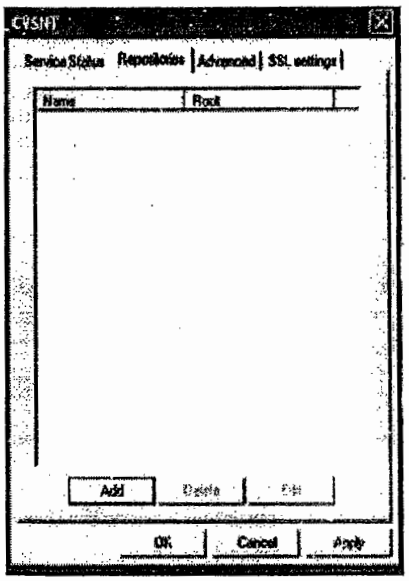
Creating The Repository



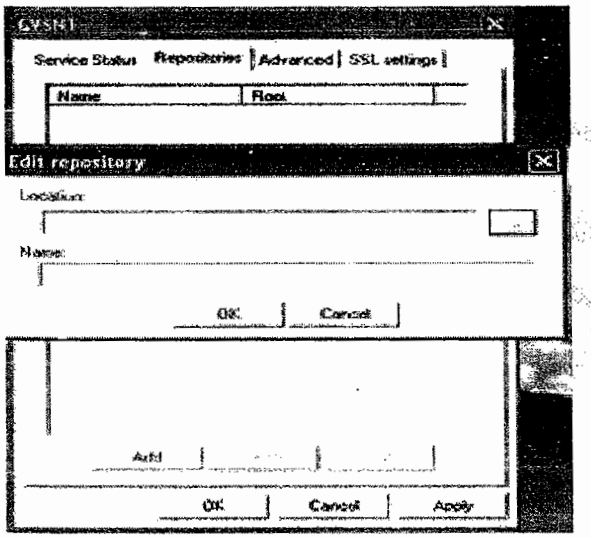
Select service controlpanel



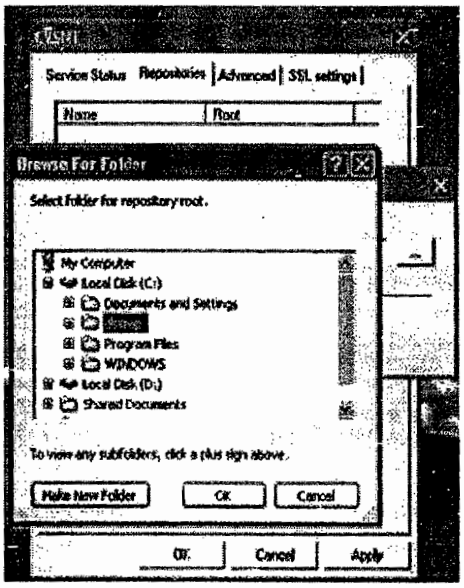
Start the CVS Service and CVS Lock Service and Go to Repositories tab

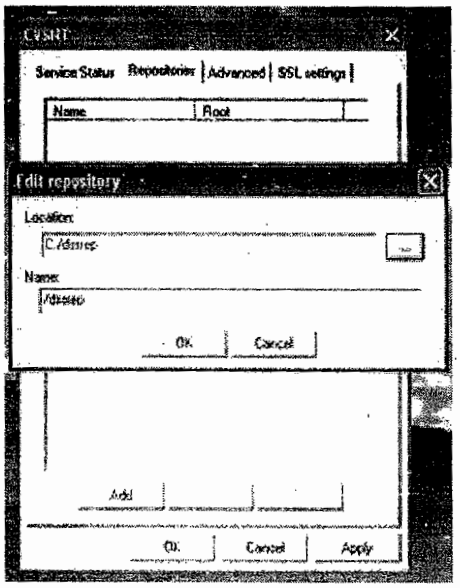


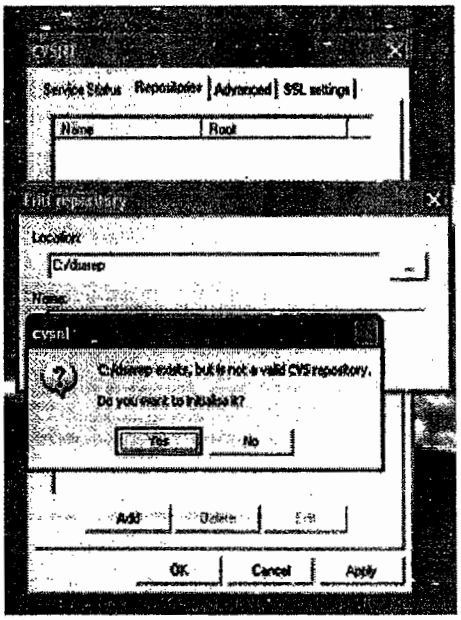
Click on Add Button



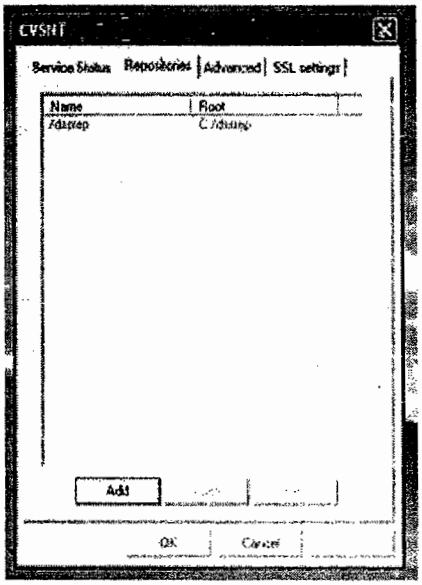
Select the repository Location and Logical Name


Service Status
Repositories
Advanced
SSL settings
Name
Root
Browse For Folder
Select folder for repository root.
My Computer
Local Disk (C:)
Documents and Settings
Program Files
WINDOWS
Local Disk (D:)
Shared Documents
To view any subfolders, click a plus sign above.
Make New Folder
OK
Cancel
OK
Cancel
Apply


Service Status
Repositories
Advanced
SSL settings
Name
Root
Edit repository
Location:
Name:
OK
Cancel
Add
OK
Cancel
Apply



Click on Yes It will intialize the CVS Repository



Click on Apply

Click on Ok

The Reppository 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

Go to repositories window → new → repository location

Give details for Host
Repository path
User
Password → finish "

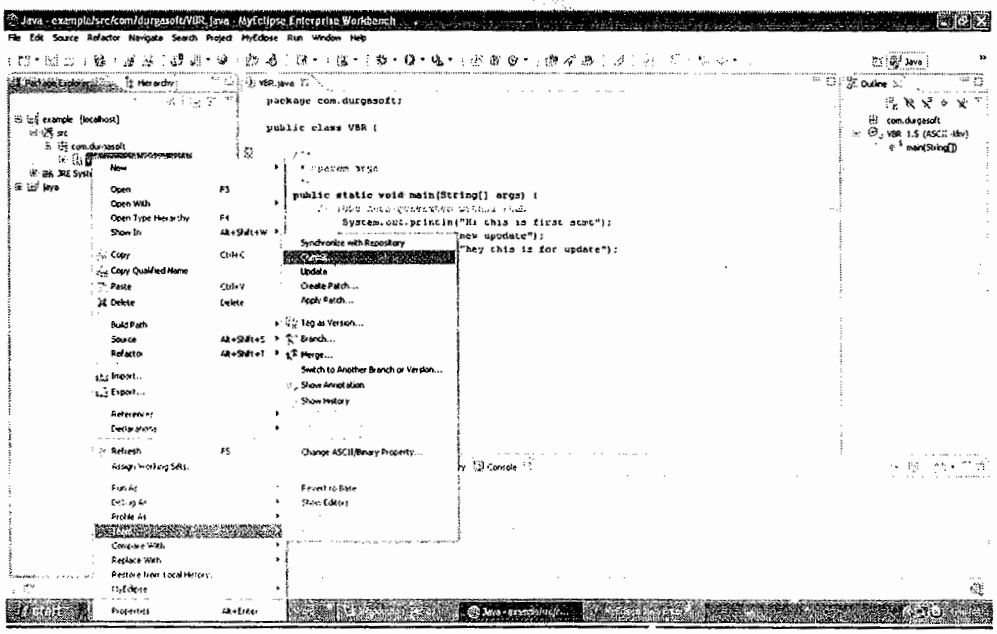
Step-3 : create a project in MyEclipse IDE.

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.

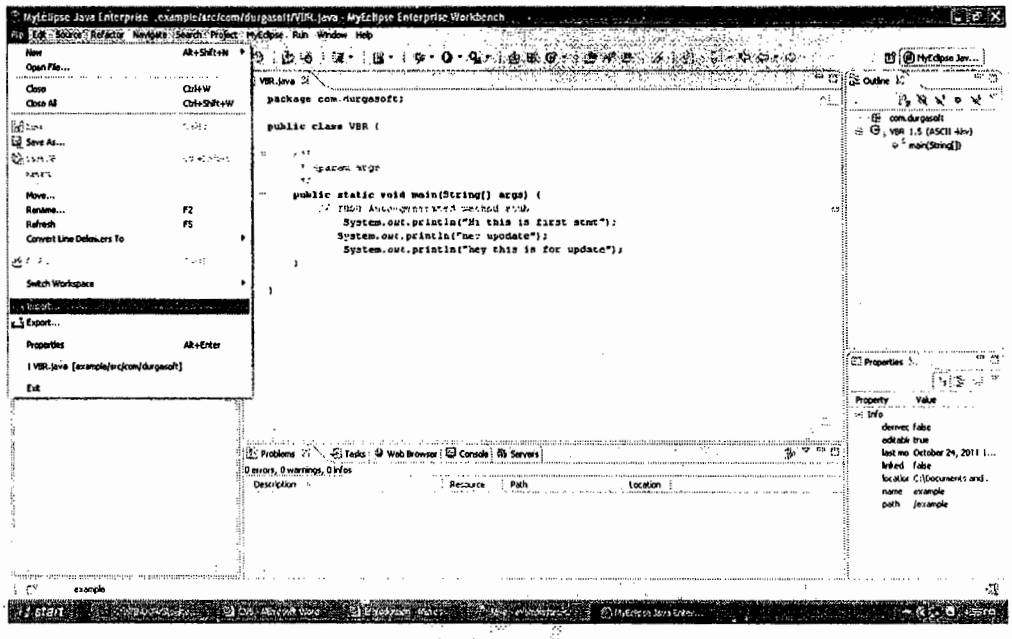
  Ex: Test .java (original file)

  Test.java 1.1 (after first modification)

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. "

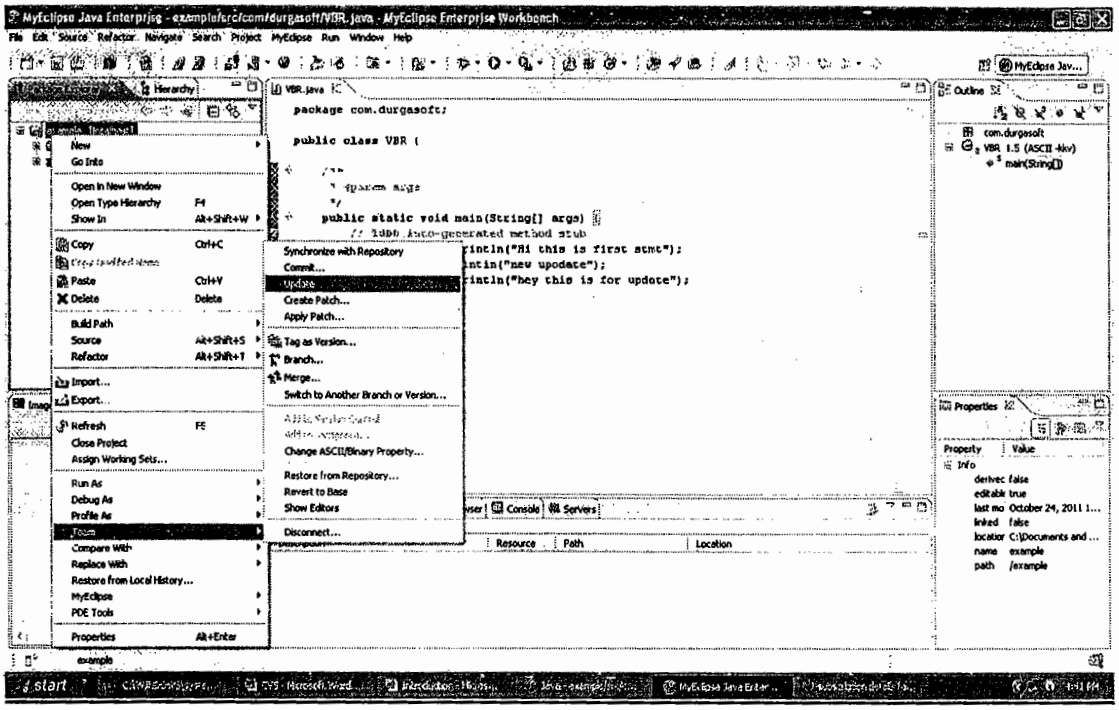


Update : This operation can be used for two purposes.

i) To get the updated version of the source file.

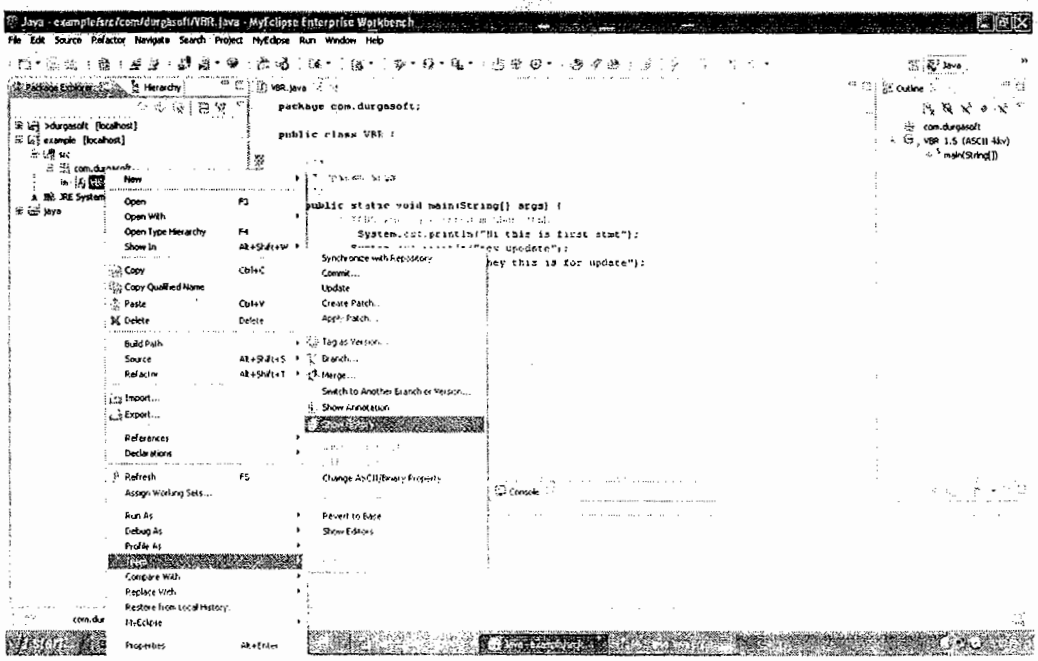
ii) To update the content of current file in cvs repository.

" **Right click on file → team → update.** "



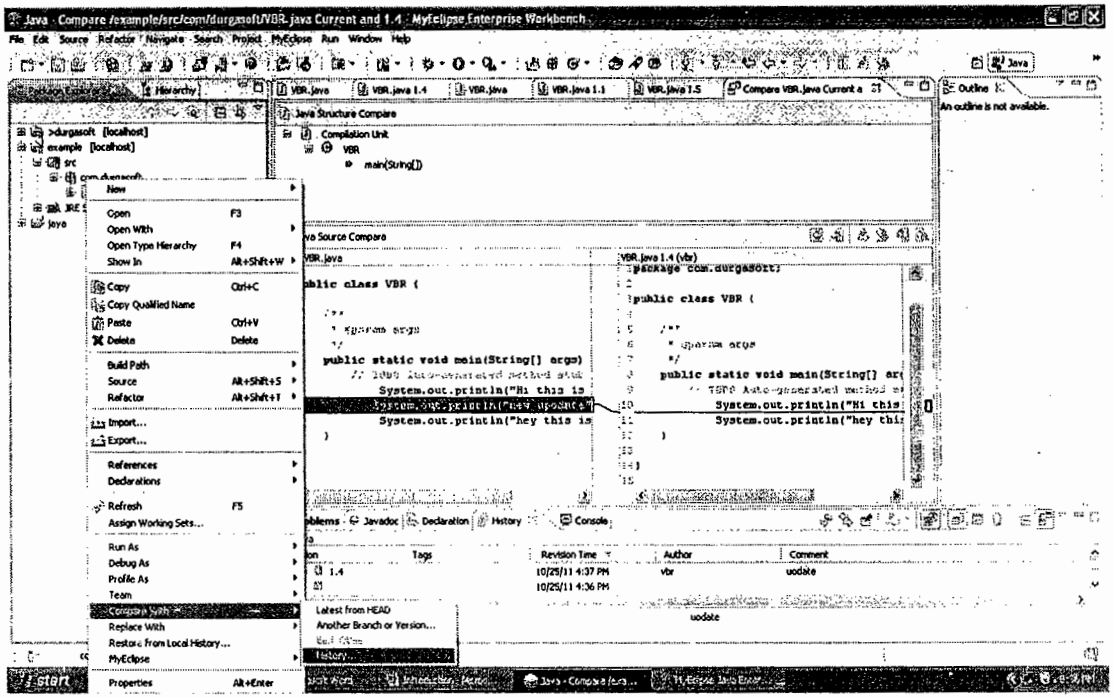
History : To replace current file content with one of the existing old version of the cvs repository .

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



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

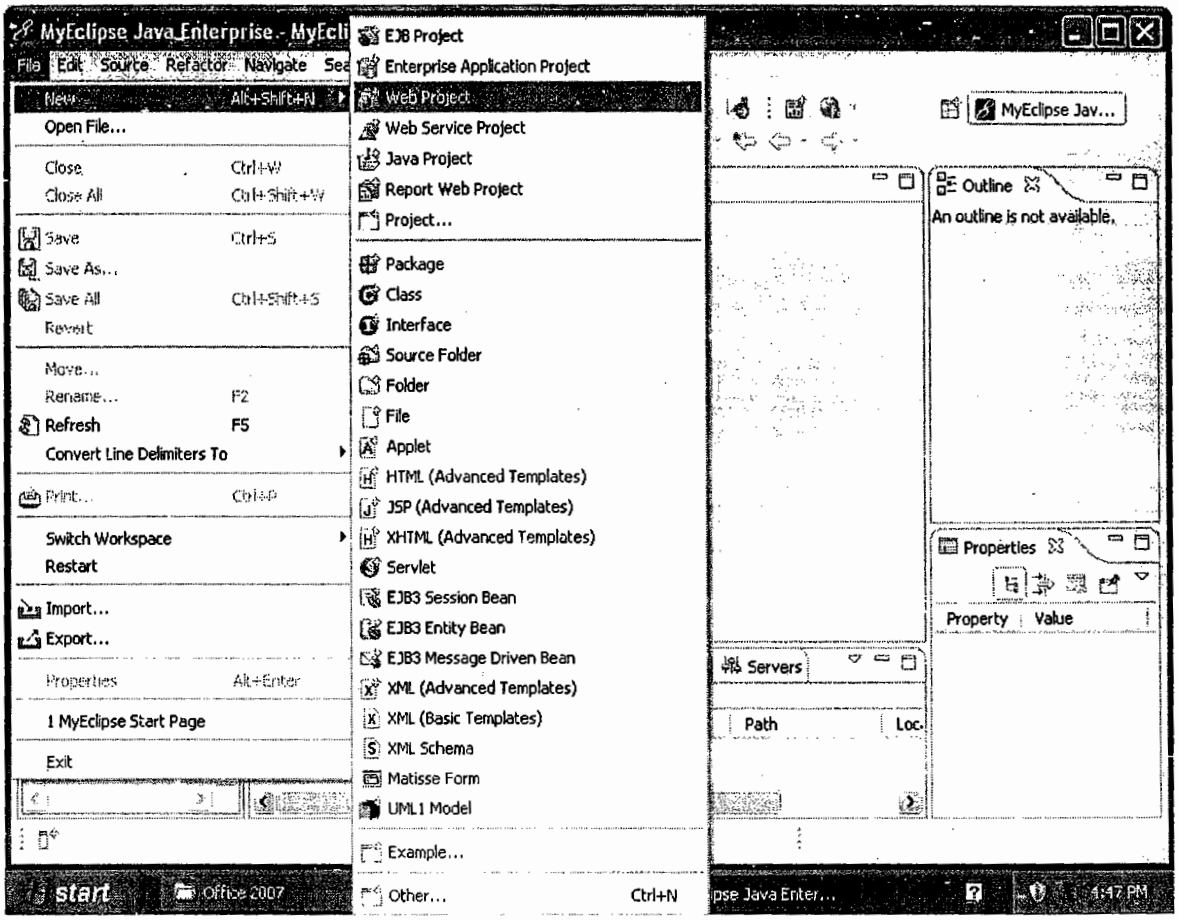
" Right click on source file → compare with → History → Go to history window select a version for compare "

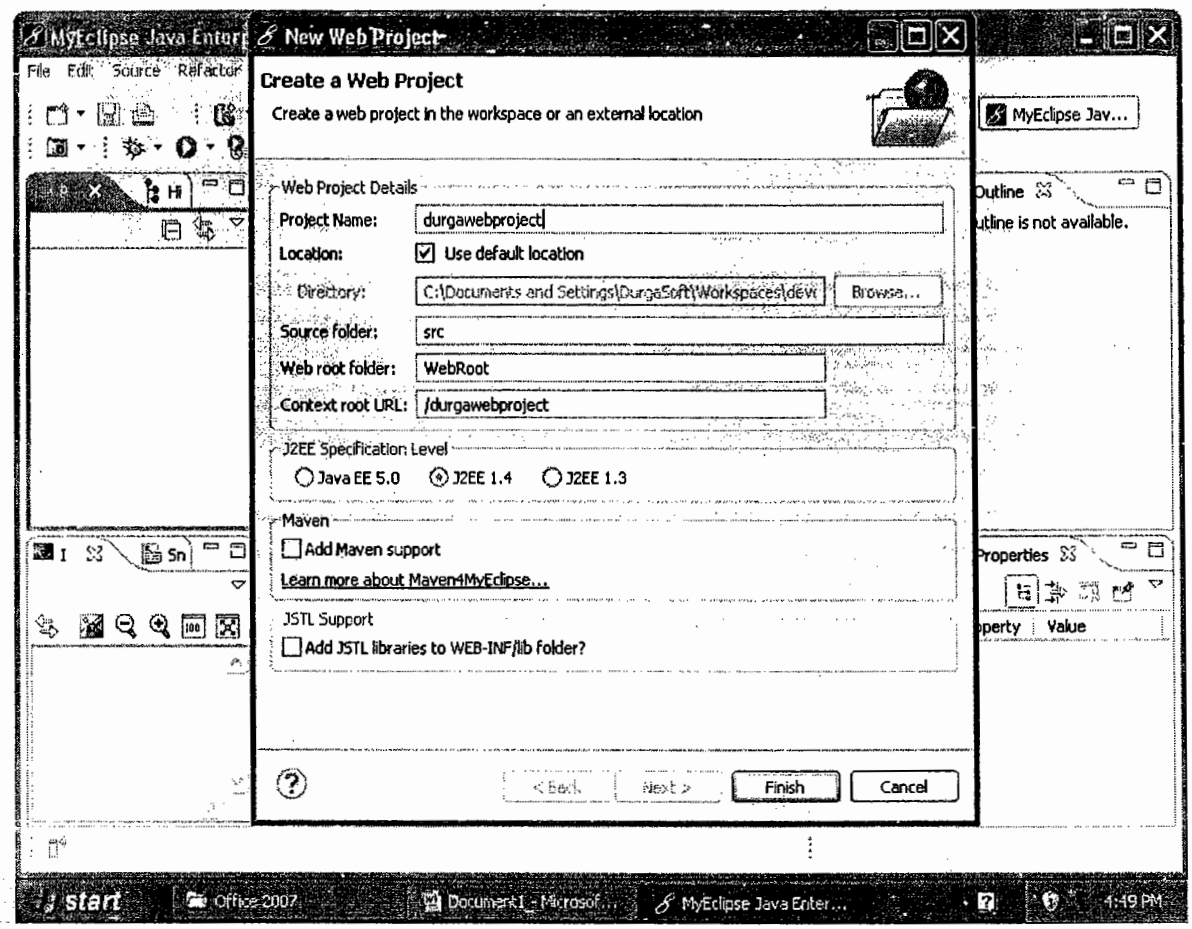


Steps to develop web application using cvs tool:

Step-1

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

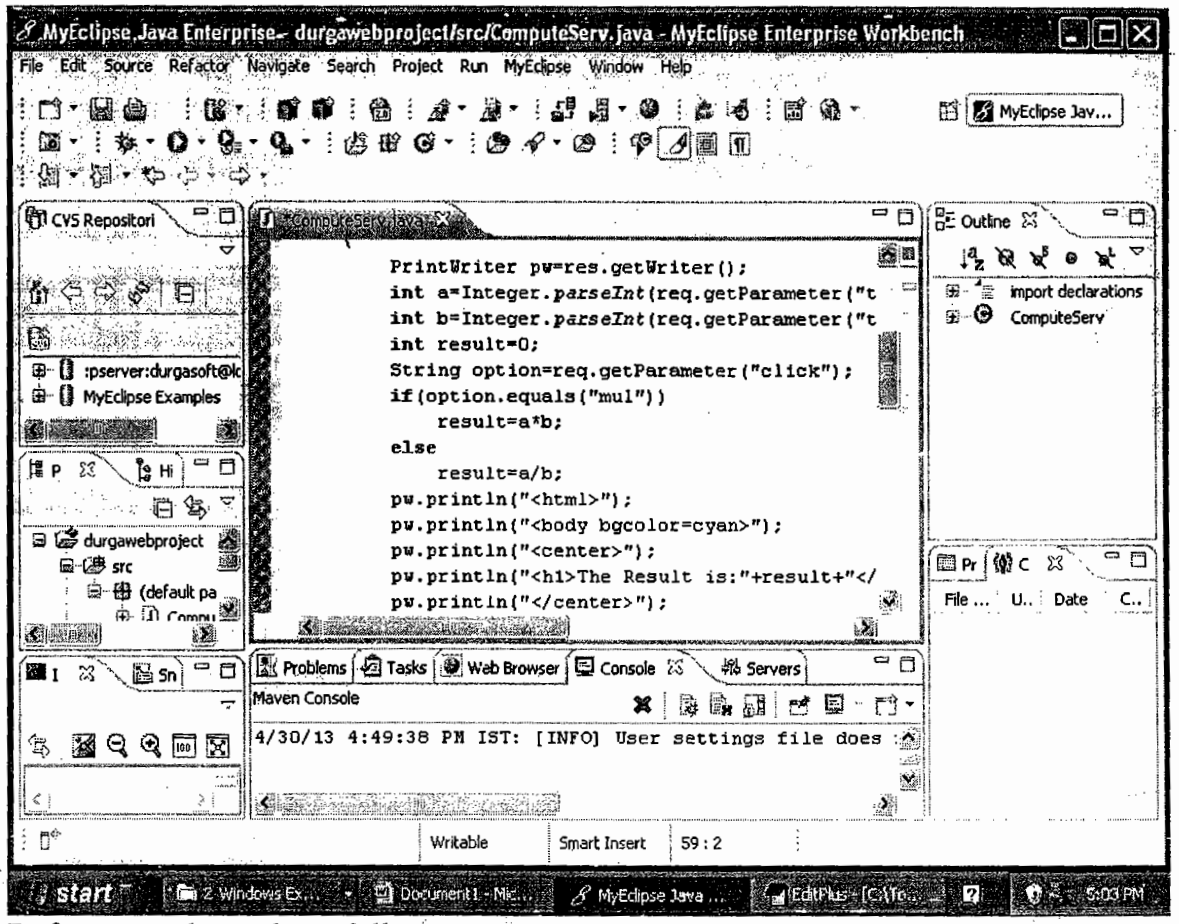

MyEclipse Java Enterprise - MyEcli
File Edit Source Refactor Navigate Sea
New... Alt+Shift+N
Open File...
Close Ctrl+W
Close All Ctrl+Shift+W
Save Ctrl+S
Save As...
Save All Ctrl+Shift+S
Revert
Move...
Rename... F2
Refresh F5
Convert Line Delimiters To
Print... Ctrl+P
Switch Workspace
Restart
Import...
Export...
Properties Alt+Enter
1 MyEclipse Start Page
Exit
EJB Project
Enterprise Application Project
Web Project
Web Service Project
Java Project
Report Web Project
Project...
Package
Class
Interface
Source Folder
Folder
File
Applet
HTML (Advanced Templates)
JSP (Advanced Templates)
XHTML (Advanced Templates)
Servlet
EJB3 Session Bean
EJB3 Entity Bean
EJB3 Message Driven Bean
XML (Advanced Templates)
XML (Basic Templates)
XML Schema
Matisse Form
UML1 Model
Example...
Other... Ctrl+N
MyEclipse Jav...
Outline
An outline is not available.
Properties
Property Value
Servers
Path Loc.
start
Office 2007
pse Java Enter...
1:47 PM



Step-2

->Develop the web resources as followes.

→first of all develope the servlet in that servlet implement the some business logic.after that implementing the presentation pages.



Refer to servlet code as follows

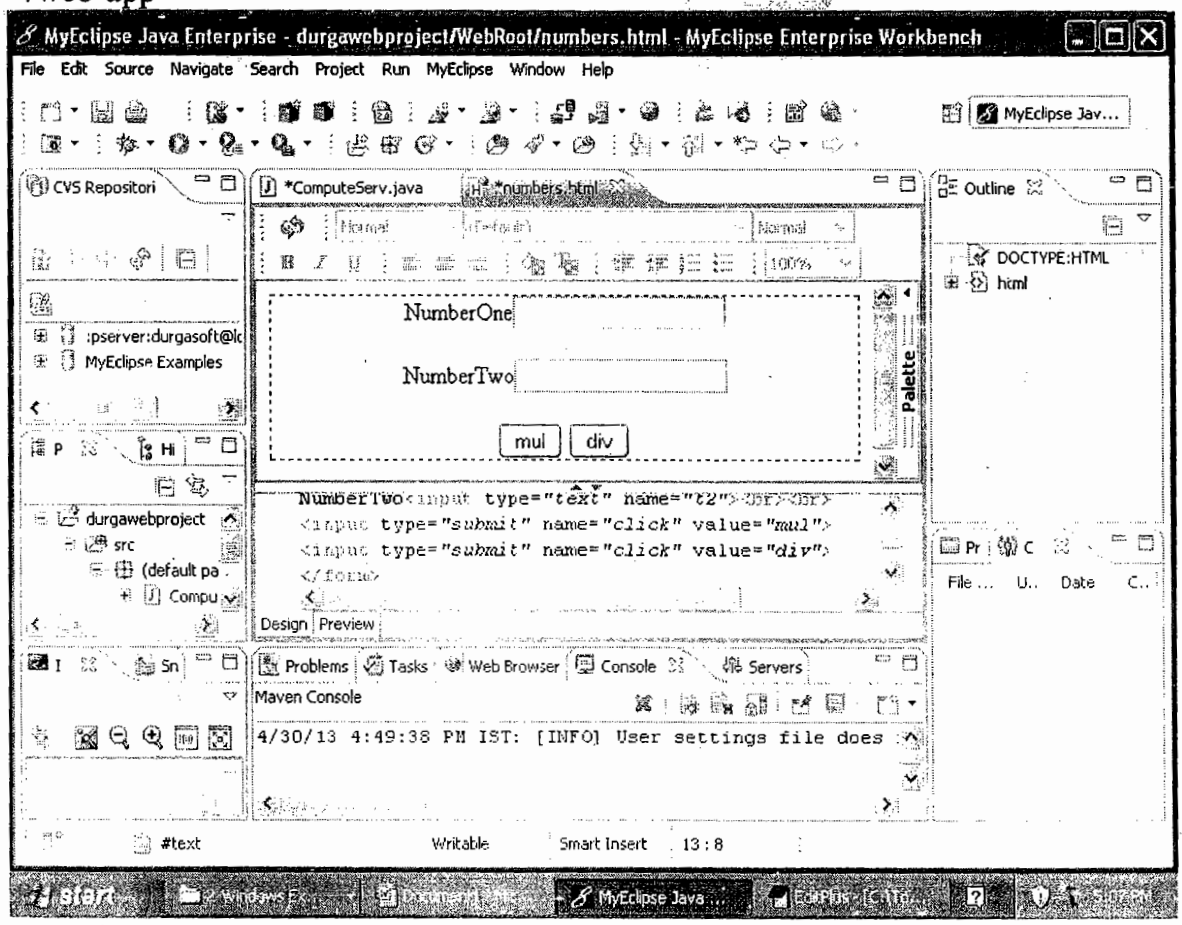
```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class ComputeServ extends HttpServlet
{
	public void doGet(HttpServletRequest req,HttpServletResponse
res)throws ServletException,IOException
	{
	System.out.println("your request is get==========!");
		process(req,res);
	}
	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"));
```

```
                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>");
```

```
            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.

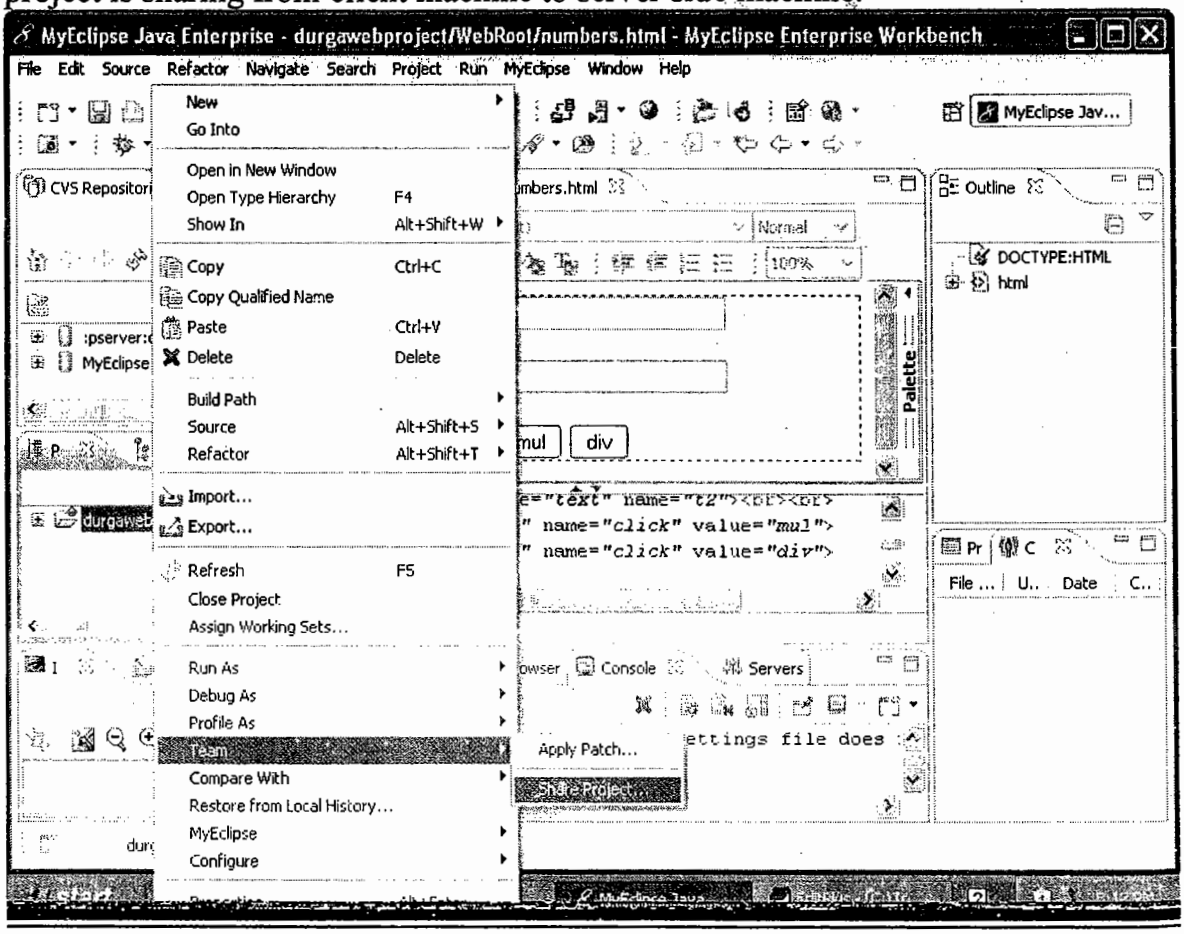
```
<html>
```

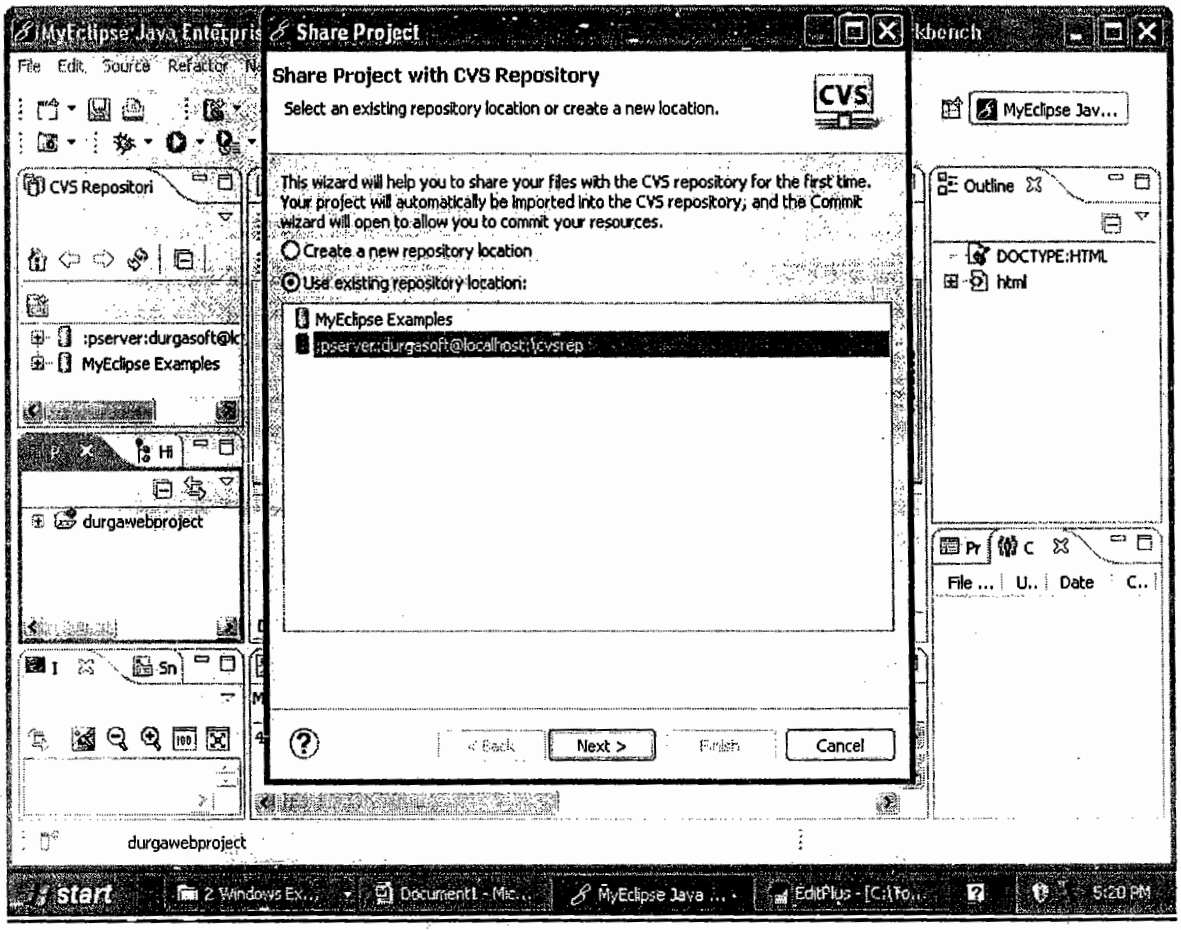
```
<body bgcolor="yellow">
<center>
<form action="./compute" method="post">
NumberOne<input type="text" name="t1"><br><br>
NumberTwo<input type="text" name="t2"><br><br>
<input type="submit" name="click" value="mul">
<input type="submit" name="click" value="div">
</form>
</center>
</body>
</html>
```

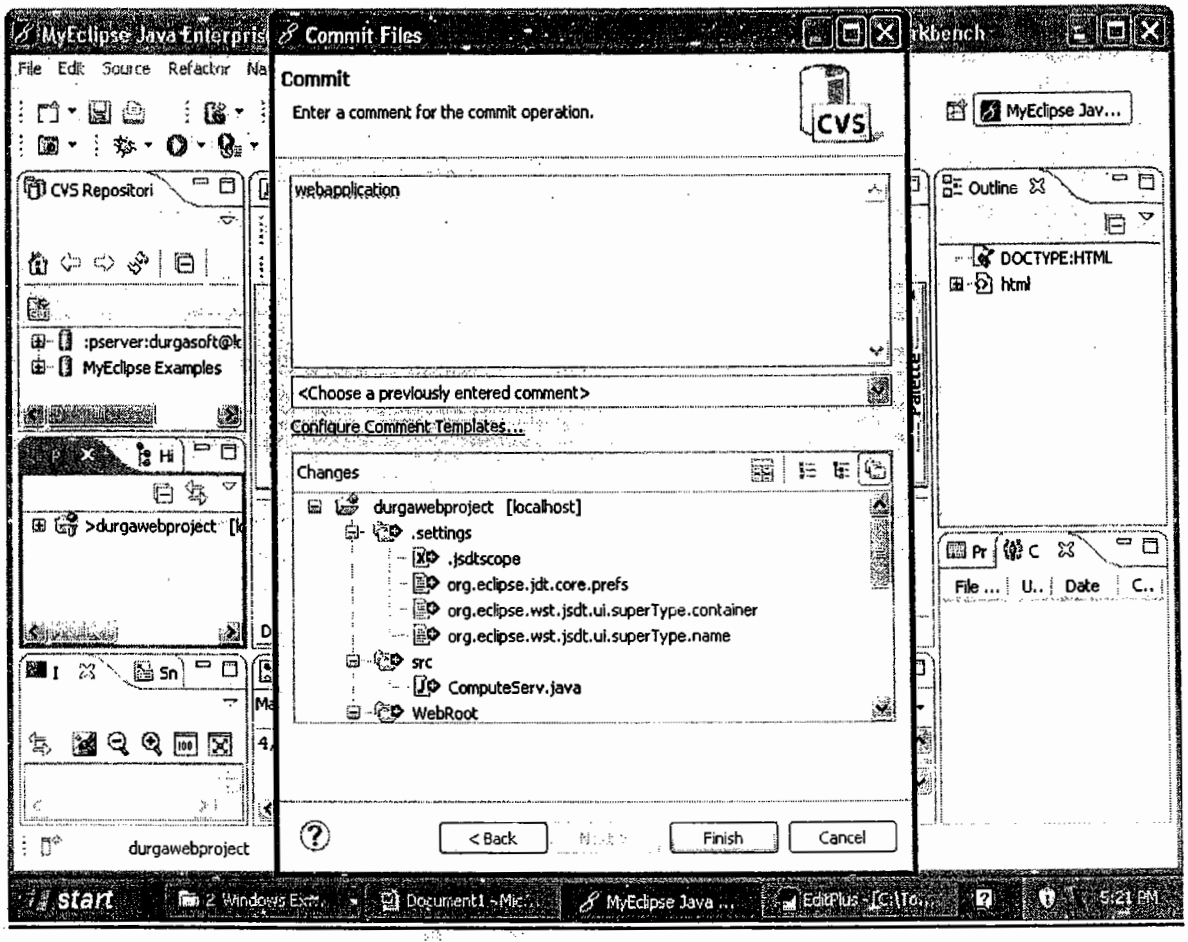
Step-3

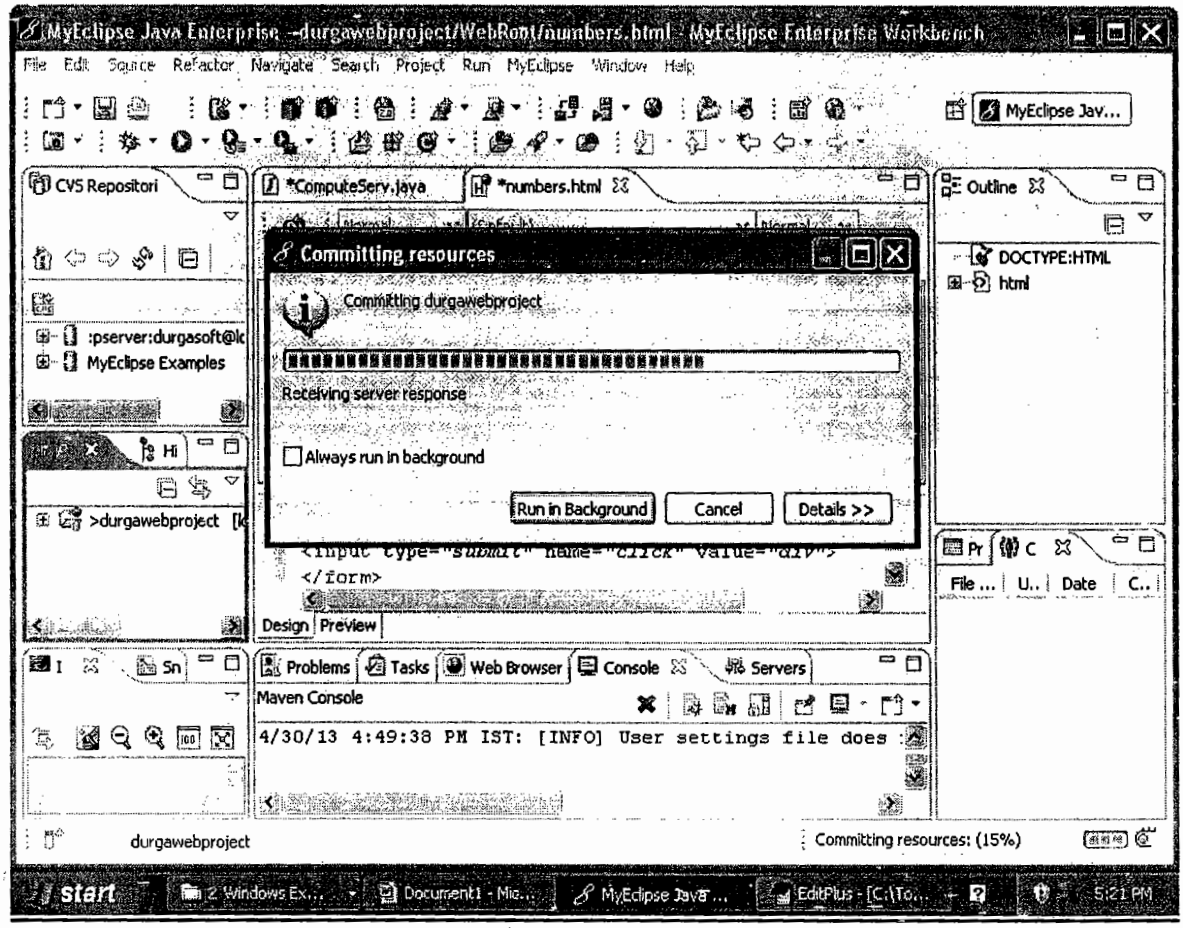
->now share the project from client machine to repository location.

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



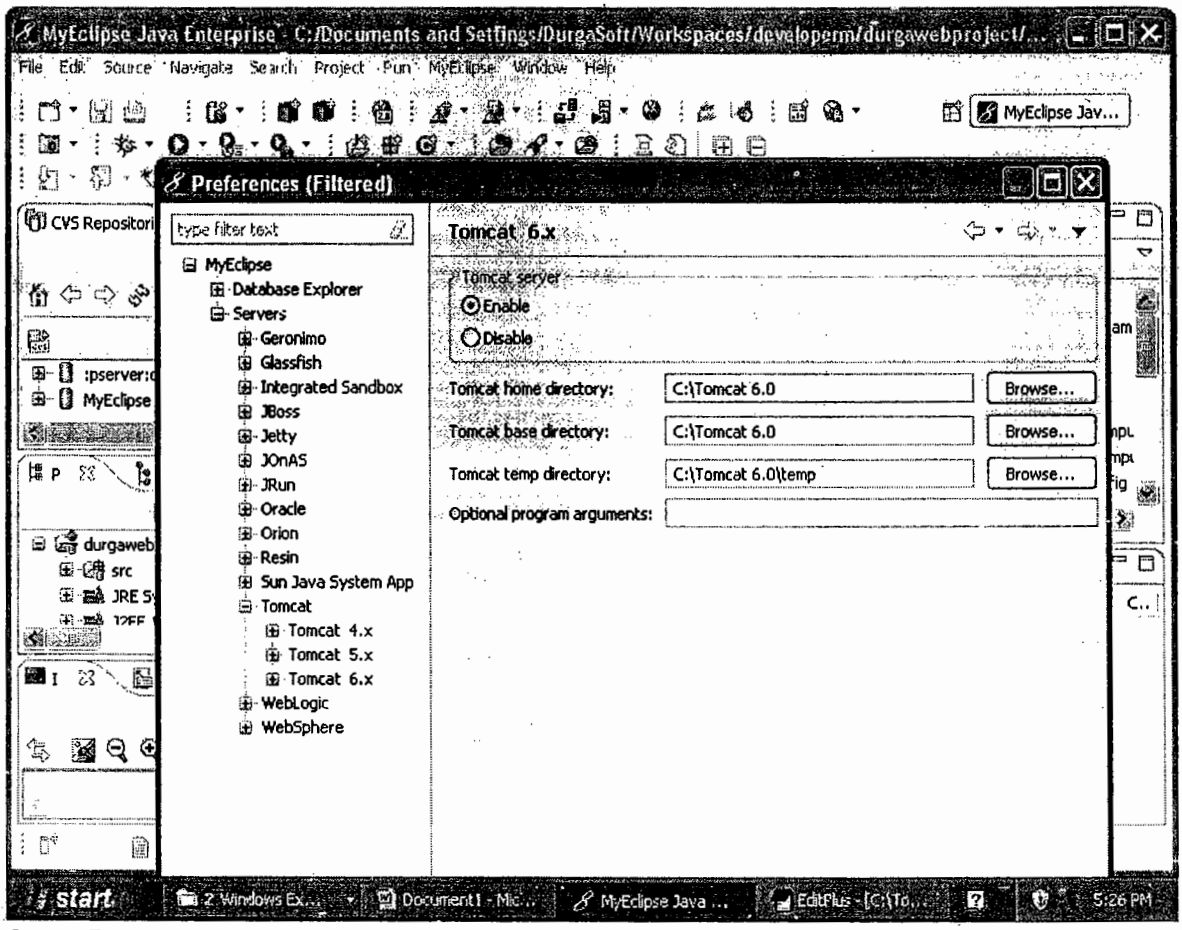

Share Project
Share Project with CVS Repository
Select an existing repository location or create a new location.
CVS
This wizard will help you to share your files with the CVS repository for the first time. Your project will automatically be imported into the CVS repository, and the Commit wizard will open to allow you to commit your resources.
Create a new repository location
Use existing repository location:
MyEclipse Examples
:pserver:durgasoft@localhost:/cvsrep
< Back
Next >
Finish
Cancel
Outline
DOCTYPE:HTML
html
durgawebproject
start


Commit Files
Commit
Enter a comment for the commit operation.
CVS
webapplication
<Choose a previously entered comment>
Configure Comment Templates...
Changes
durgawebproject [localhost]
.settings
.jsdtscope
org.eclipse.jdt.core.prefs
org.eclipse.wst.jsdt.ui.superType.container
org.eclipse.wst.jsdt.ui.superType.name
src
ComputeServ.java
WebRoot
< Back
Finish
Cancel
CVS Repositori
:pserver:durgasoft@k
MyEclipse Examples
>durgawebproject
durgawebproject
Outline
DOCTYPE:HTML
html
MyEclipse Jav...
start
MyEclipse Java ...



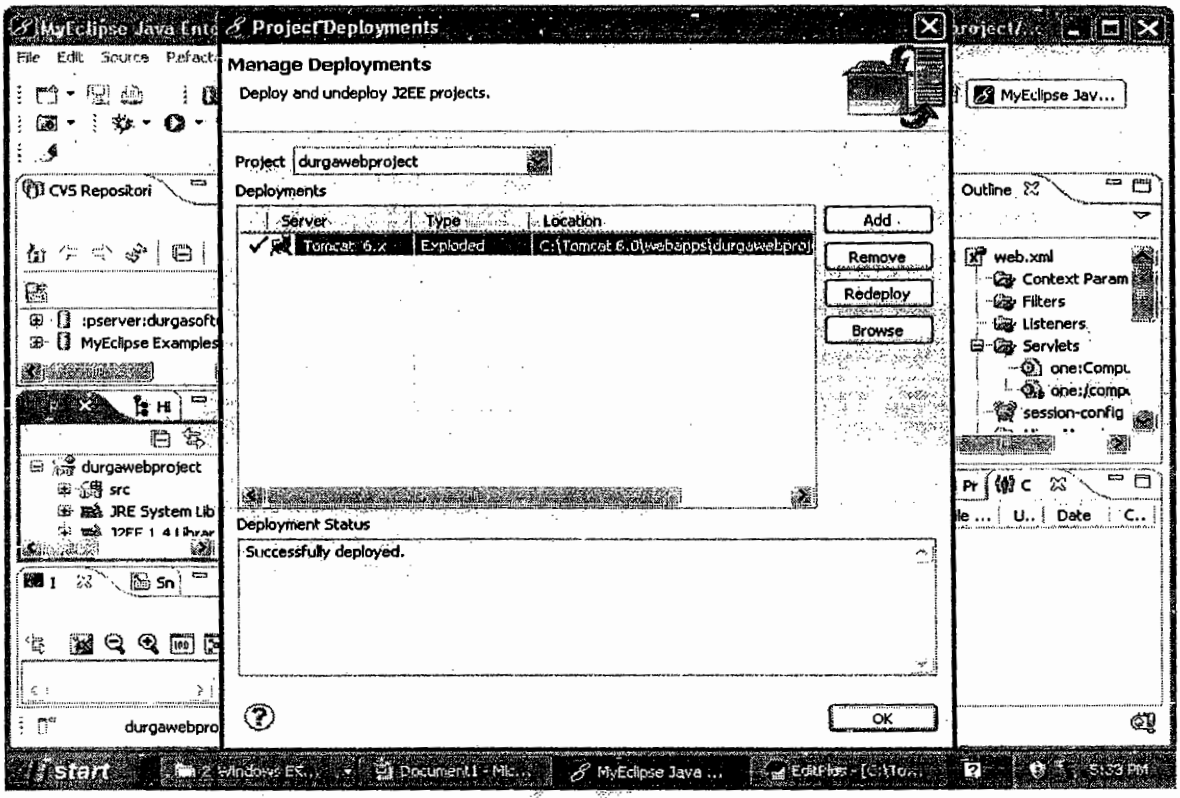
Step-4

Configure the server as follows.



Step-5

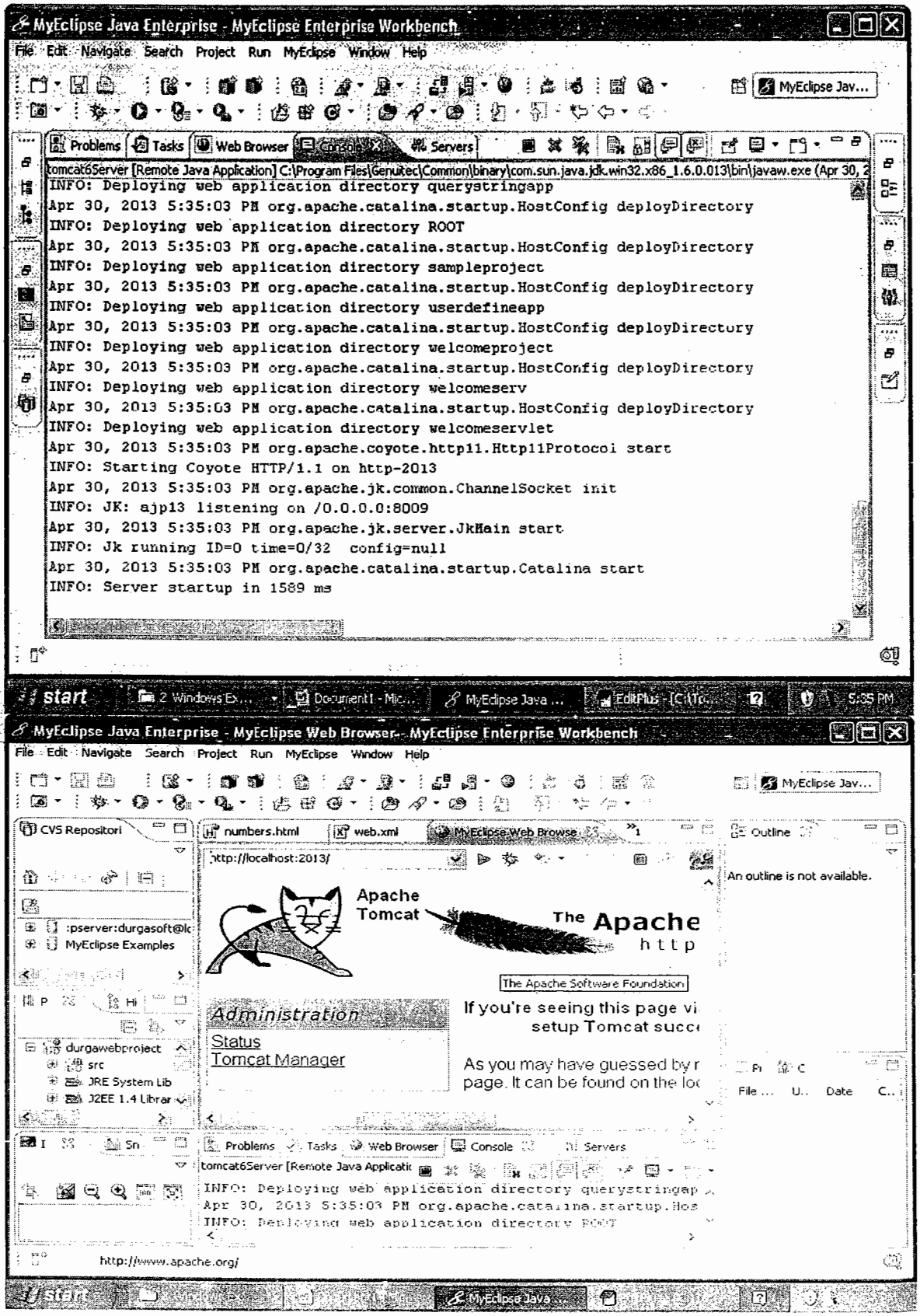
Deploye the web application in tomcat server.

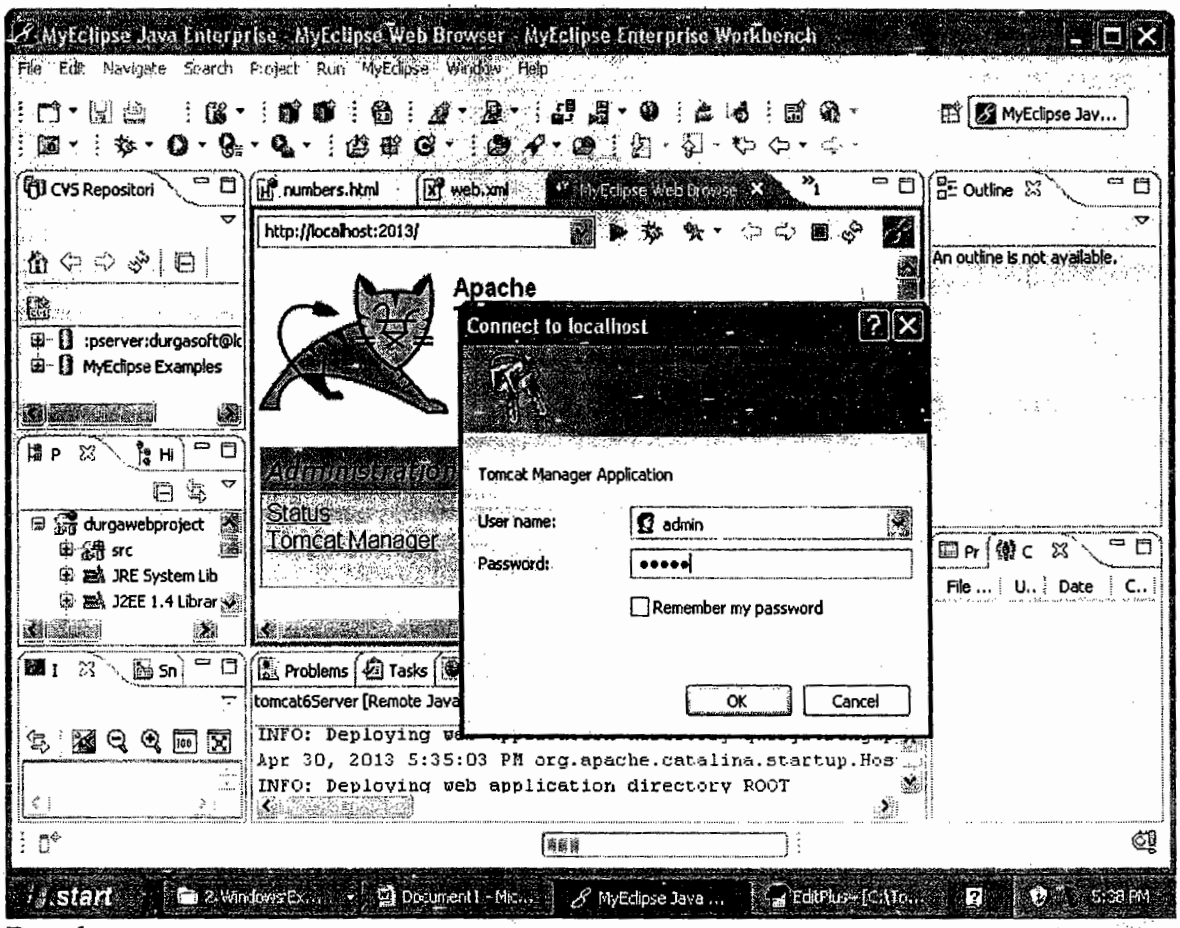


Step-6

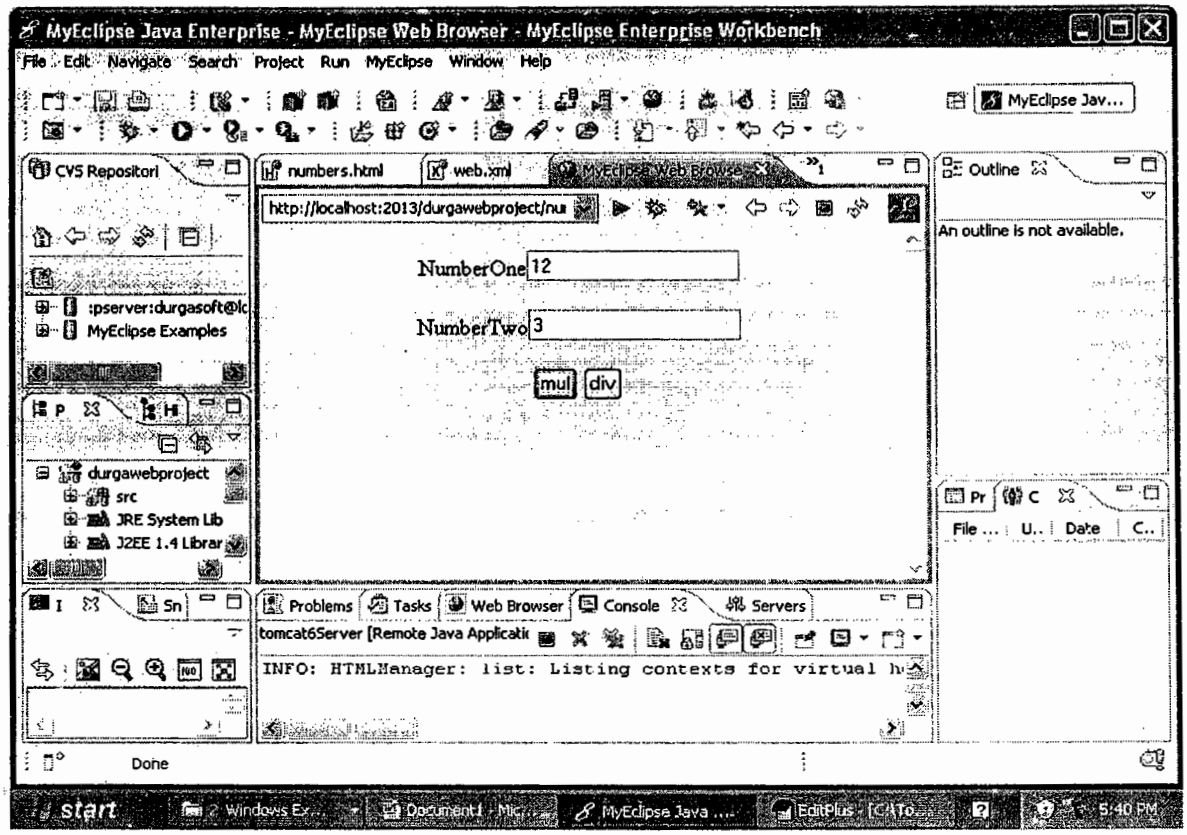
Now start the server and testing the application.

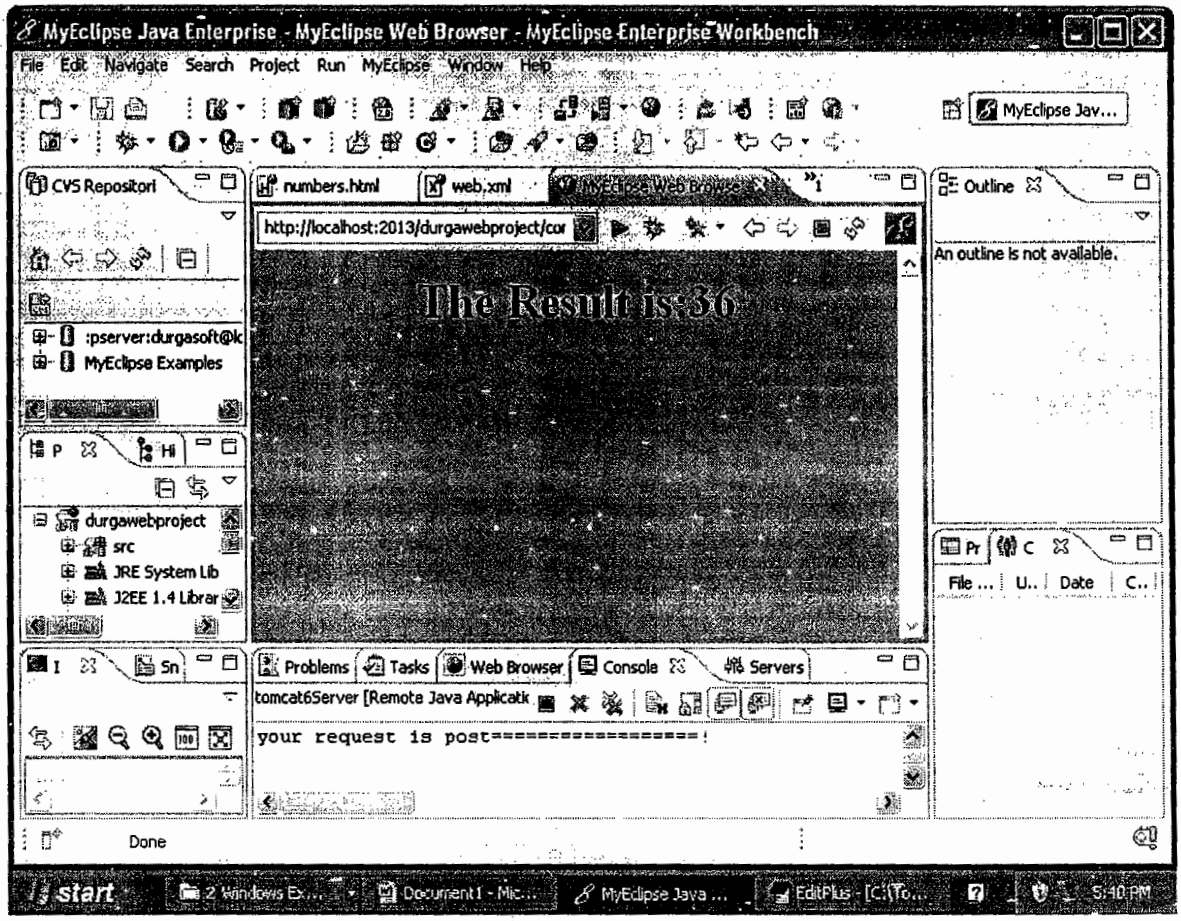
Now open the internet explorer and make a request to web application.


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\Common\binary\com.sun.java.jdk.win32.x86_1.6.0.013\bin\javaw.exe (Apr 30, 2
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 sampleproject
Apr 30, 2013 5:35:03 PM org.apache.catalina.startup.HostConfig deployDirectory
INFO: Deploying web application directory userdefineapp
Apr 30, 2013 5:35:03 PM org.apache.catalina.startup.HostConfig deployDirectory
INFO: Deploying web application directory welcomeproject
Apr 30, 2013 5:35:03 PM org.apache.catalina.startup.HostConfig deployDirectory
INFO: Deploying web application directory welcomeserv
Apr 30, 2013 5:35:03 PM org.apache.catalina.startup.HostConfig deployDirectory
INFO: Deploying web application directory welcomeservlet
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: ajp13 listening on /0.0.0.0:8009
Apr 30, 2013 5:35:03 PM org.apache.jk.server.JkMain start
INFO: Jk running ID=0 time=0/32 config=null
Apr 30, 2013 5:35:03 PM org.apache.catalina.startup.Catalina start
INFO: Server startup in 1589 ms
start
MyEclipse Java Enterprise - MyEclipse Web Browser - MyEclipse Enterprise Workbench
CVS Repositori
numbers.html
web.xml
MyEclipse Web Browse
http://localhost:2013/
Apache Tomcat
The Apache
The Apache Software Foundation
Administration
Status
Tomcat Manager
Outline
An outline is not available.
MyEclipse Examples
durgawebproject
src
JRE System Lib
Problems Tasks Web Browser Console Servers
http://www.apache.org/


MyEclipse Java Enterprise - MyEclipse Web Browser - MyEclipse Enterprise Workbench
File Edit Navigate Search Project Run MyEclipse Window Help
MyEclipse Jav...
CVS Repositori
numbers.html
web.xml
MyEclipse Web Browse
Outline
http://localhost:2013/
An outline is not available.
:pserver:durgasoft@lc
MyEclipse Examples
Apache
Connect to localhost
Tomcat Manager Application
User name:
admin
Password:
Remember my password
OK
Cancel
Administration
Status
Tomcat Manager
durgawebproject
src
JRE System Lib
J2EE 1.4 Librar
Problems
Tasks
tomcat6Server [Remote Java
INFO: Deploying
Apr 30, 2013 5:35:03 PM org.apache.catalina.startup.Hos
INFO: Deploying web application directory ROOT
File ...
U..
Date
C..
start
2 Windows Ex...
Document1 - Mic...
MyEclipse Java ...
EditPlus - [C:\To...
5:38 PM

Result


MyEclipse Java Enterprise - MyEclipse Web Browser - MyEclipse Enterprise Workbench
File Edit Navigate Search Project Run MyEclipse Window Help
CVS Repositori
numbers.html
web.xml
http://localhost:2013/durgawebproject/nu
NumberOne 12
NumberTwo 3
mul
div
Outline
An outline is not available.
durgawebproject
src
JRE System Lib
J2EE 1.4 Librar
Problems
Tasks
Web Browser
Console
Servers
tomcat6Server [Remote Java Applicati
INFO: HTMLManager: list: Listing contexts for virtual h
Done
start
5:40 PM


MyEclipse Java Enterprise - MyEclipse Web Browser - MyEclipse Enterprise Workbench
File Edit Navigate Search Project Run MyEclipse Window Help
MyEclipse Jav...
CVS Repositori
numbers.html
web.xml
Outline
An outline is not available.
http://localhost:2013/durgawebproject/cor
The Result is 36
:pserver:durgasoft@lc
MyEclipse Examples
durgawebproject
src
JRE System Lib
J2EE 1.4 Librar
File ...
U..
Date
C..
Problems
Tasks
Web Browser
Console
Servers
tomcat6Server [Remote Java Applicatic
your request is post==========!
Done
start
2 Windows Ex...
Document1 - Mic...
MyEclipse Java ...
EditPlus - [C:\To...
5:40 PM

Report indicates the process of data retrieving and displaying that data in diff formats having pagenation support. Every report is useful to analyze the data of the application eg:- Sales Report, Bank Account statement, progress Report & etc.

Various Reporting Tools
1) Jasper Reports
2) I Reports
3) Accute Reports
4) Crystal Reports
5) Data Reports, & etc.

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:

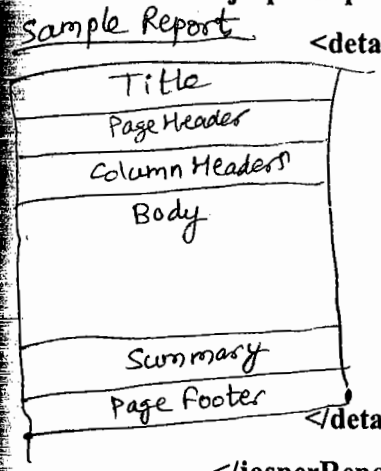
Jasper Reports organizes data retrieved from a data source according to a report-design defined in a JRXML file. In order to fill a report with data, the report-design must be compiled first.

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.

Eg:

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



Jasper Reports
version : 3.7.x
→ type : java based Reporting tool
→ vendor : Jasper Soft
→ Open Source
→ To download s/w : Download jasperreports-3.7.5-project.zip file from www.jaspersoft.com
→ jar file : jasperreports-3.7.5.jar

- **<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

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

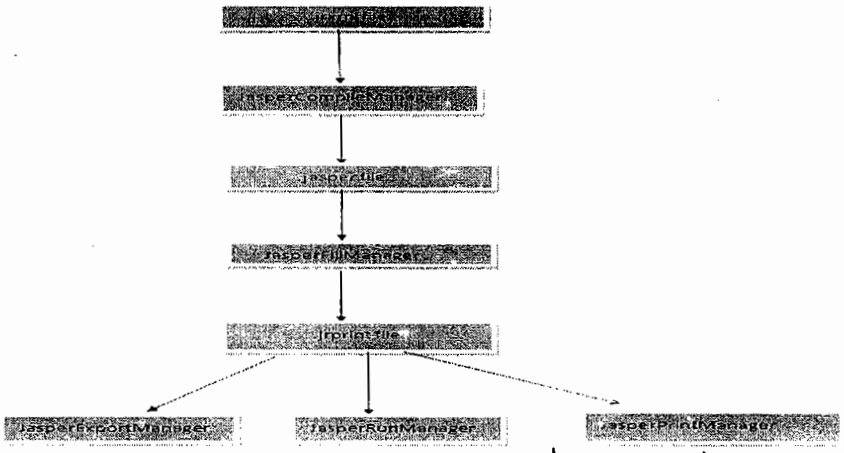
These classes represent a façade to the Jasper Reports engine. They have various static methods that simplify the access to the API functionality and can be used to compile an JRXML report design, to fill a report, to print it, or to export to other document formats (PDF, HTML, XML).

c) <u>View, Print and Export Reports:</u>

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.



For More examples refer applications given in page no:- 281

<u>Parameters:</u>

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"/>
```

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}
```

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}
```

Data Source:

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 implemetations 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"/>
```

Expressions:

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>
$F{FirstName} + " " + $F{LastName}
</textFieldExpression>
```

The expression can be even more complex:

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

</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>
```

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:

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>
```

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.

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

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:

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:

- **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>**

```
                    1. <reportElement x="20" y="0" width="200"
                       height="40"    forecolor="#FF3300"
                       backcolor="#FFFFFF"/>
                    2. <text><![CDATA[Welcome To
                       DurgaSoftwareSollutions]]></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. }
```

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" pagcWidth="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"/>
```

```
                    b. <textElement textAlignment="Center">
                        i. <font fontName="Times New Roman"
                           pdfFontName="Times-Roman"/>
                    c. </textElement>
                    d. <textFieldExpression
                       class="java.lang.String">$P{ReportTitle}</text
                       FieldExpression>
                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. <font isBold="false"/>
                    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. <font isBold="false"/>
                    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. <font isBold="false"/>
                    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>
```

```
                    i. <font isBold="false"/>
                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. <font isBold="false"/>
                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. <font isBold="false"/>
                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. <font size="14"/>
                    iii. </textElement>
                    iv. <textFieldExpression
                       class="java.math.BigDecimal"><![CDAT
                       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. <font size="14"/>
```

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[$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>

```
                    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. <font size="10" isBold="false"/>
                    iii. </textElement>
                    iv. <textFieldExpression
                       class="java.lang.String"><![CDATA[" "
                       +
                       $V{PAGE_NUMBER}]]></textFieldExpr
                       ession>
                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">
```

```
                                    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="EEEEE 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;
```

```
        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);
```

```
        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");
```

```
        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);
```

```
                }

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

```
        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();
                                }
        }
            }
        }
}
```

I Reports Tool:

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

Throught a rich and very simple to use GUI, iReport provide all the most importatnt 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 rewrited. For this reason they are two manuals for iReport. The new development direction was taken for a lot of reasons. iReport has lost a bit of efficence leaving the win32 native GUI for a clear, pure swing interface. But this is the right direction...

1Set Up To Launch iReport-3.7.5 Tool

Step1:

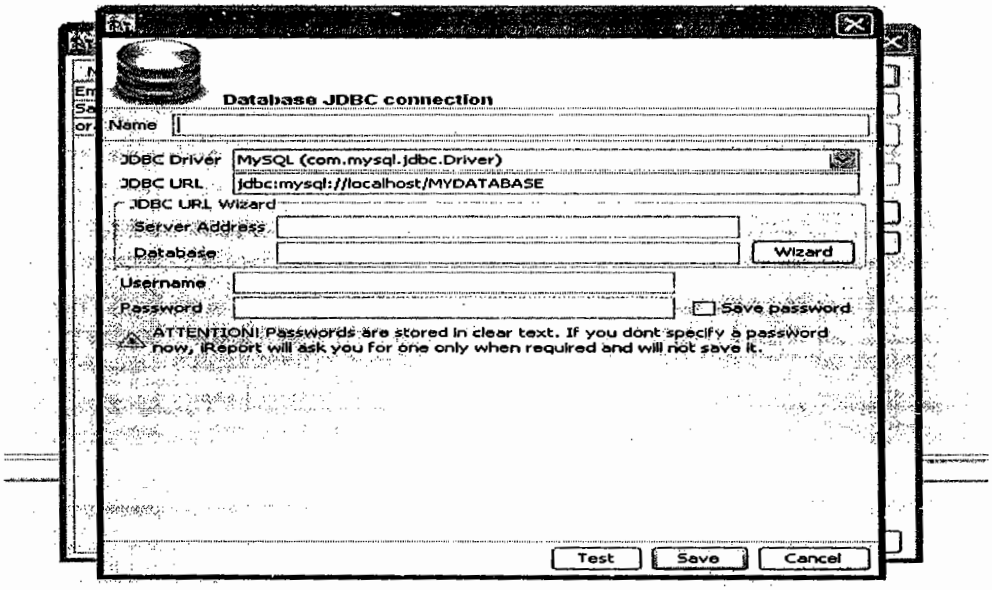
Start→Programs→Jasper Soft→iReports-3.7.5→iReport3.7.5

Step2:

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

Step3:

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



(provide the names as shown in above)
Name:orads

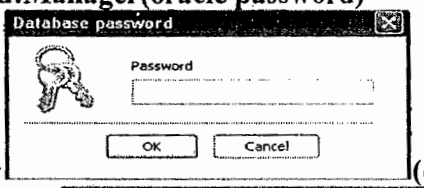
JdbcDriver:Oracle(oracle.jdbc.driver.OracleDriver)

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

ServlerAddress:localhost

UserName:system(oracle username)

Password:Manager(oracle password)



→Test→ **(enter database**

password)→ **→save→close**

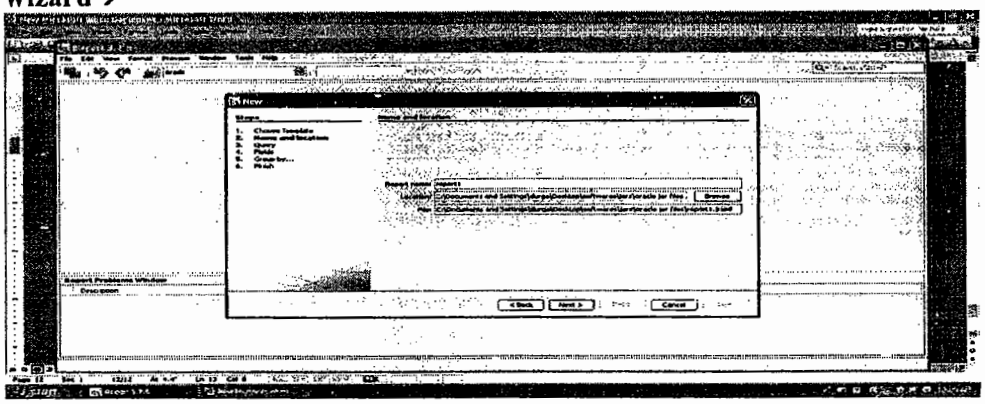
<u>Procedure to design report byusing iReport tool:</u>

<u>Step1:</u>

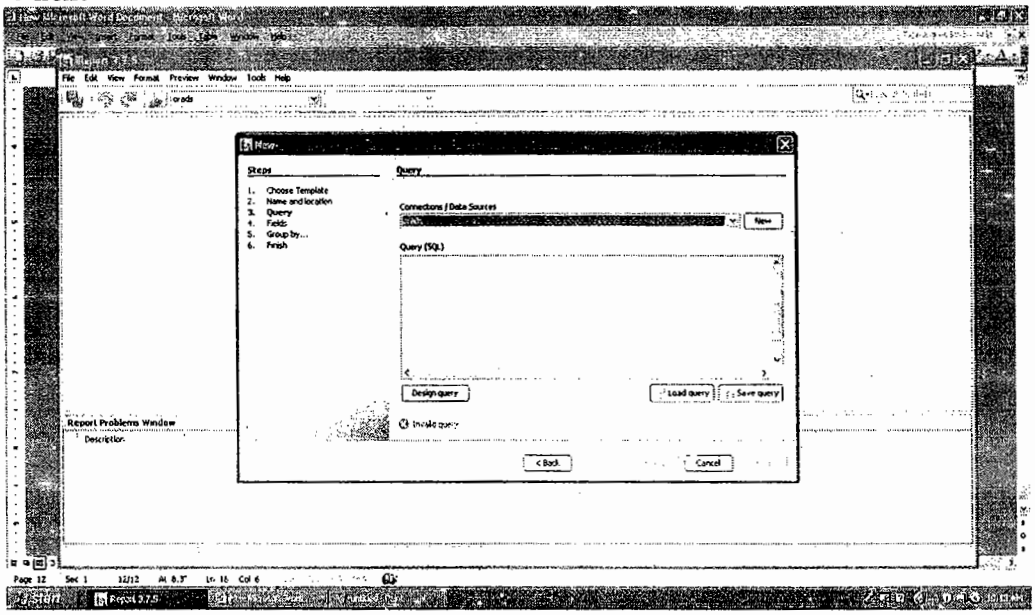
Start→Programs→Jasper Soft→iReports-3.7.5→iReport3.7.5

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

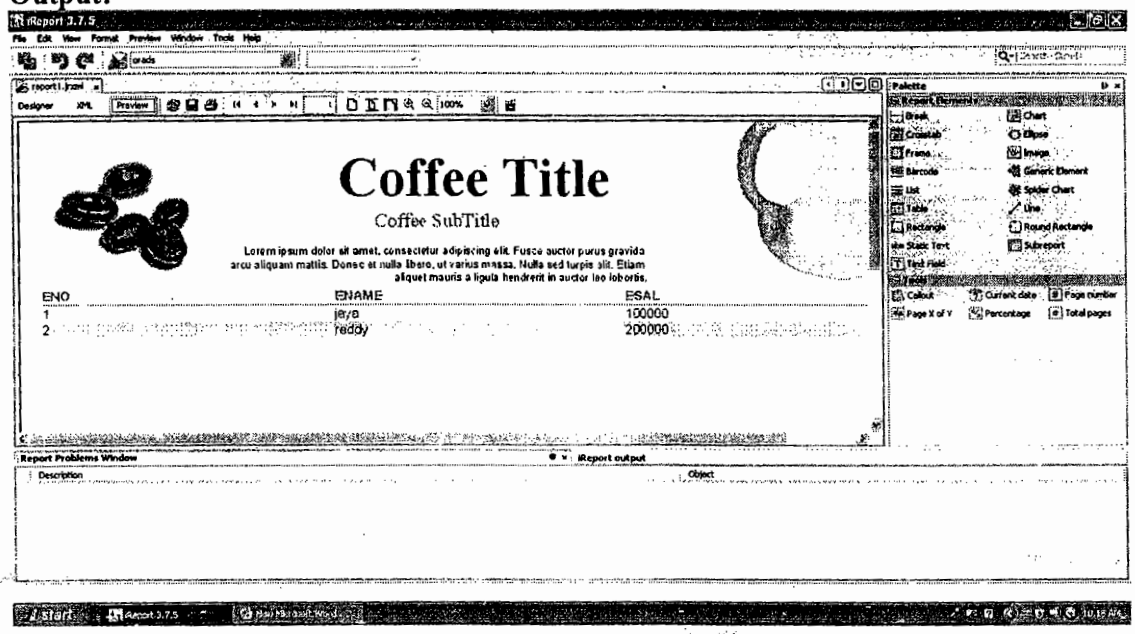


Provide the in the fields:
Connections/datasources:orads

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

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:

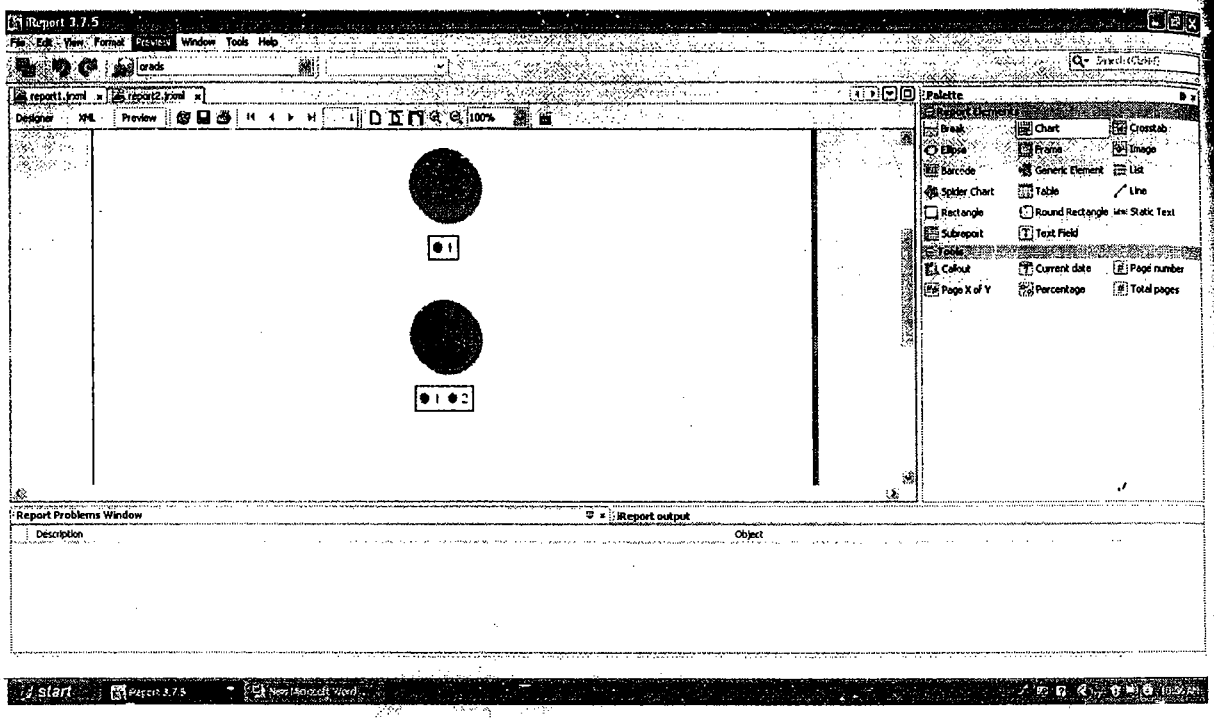
Step1: Start→Programs→Jasper Soft→iReports-3.7.5→iReport3.7.5

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

Step3:

select pallets→drag & drop chart to detail1→select Pie(any one)→ok→dataset:Main report DataSet→next→select unique idetifer: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 equalent .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:

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

ant	maven
1) Build tool	1) build tool and project management tool
2) Does not give common Directory structure for Apps	2) Gives
3) No life cycle	3) It has life cycle.
4) No plugins support	4) Supports plugins (reusability)
5) Does not support project Inheritance.	5) supports project inheritance
6) Does not download jar, plugins dynamically.	6) Downloads automatically.
7) No repositories	7) Support Repositories..

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

For installation and configuration of maven tool and basic information about maven tool refer page no :- 289

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.

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.

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>
```

<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.

- **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 use 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.

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.

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.

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

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

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

- 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.

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>
<repositories>
<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.

- **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 is 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

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

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 names 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 simplifed 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.

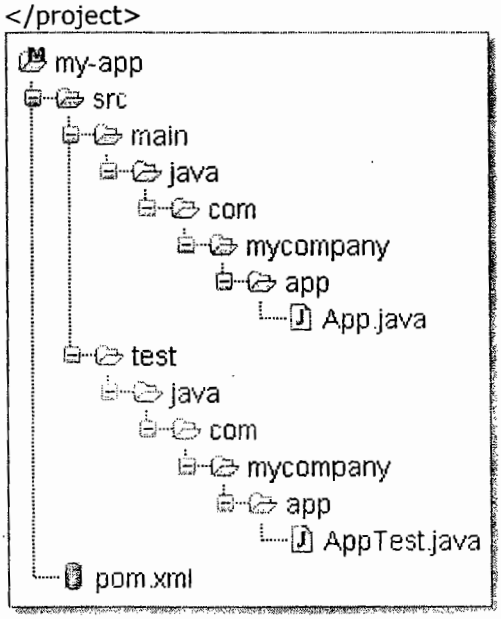
webapp	

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

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>
```

```
<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

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

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:

• 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:

```
[...]
<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

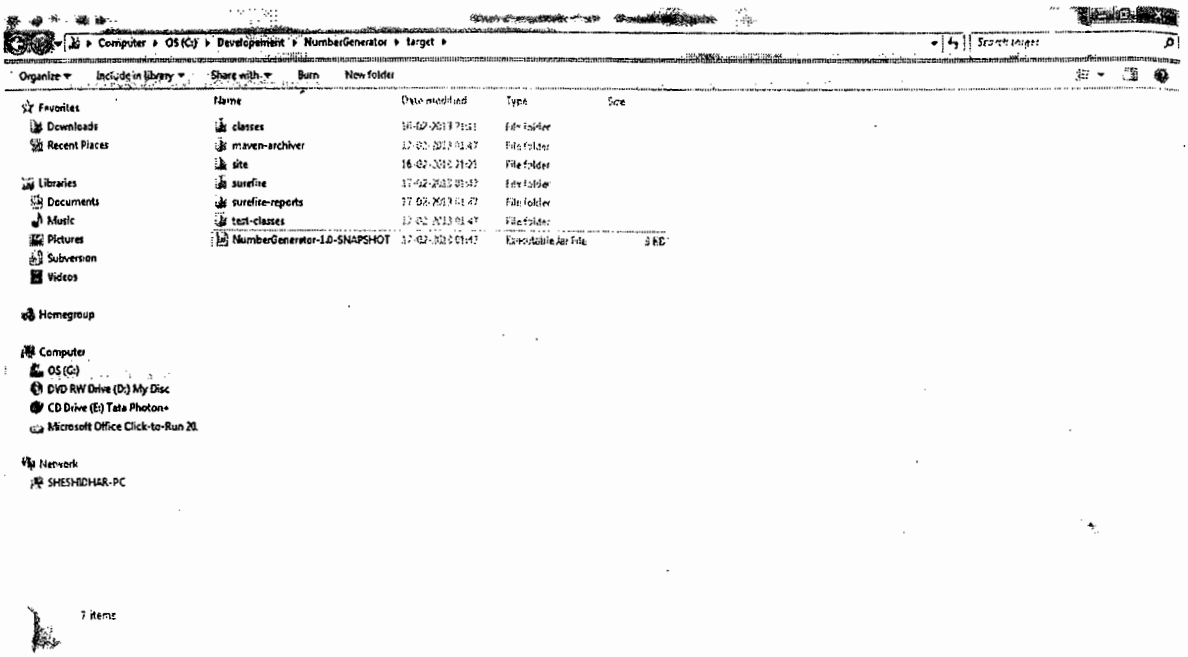
Compiling Test Sources and Running Unit Tests:

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

Packaging and Installation to Your Local Repository:

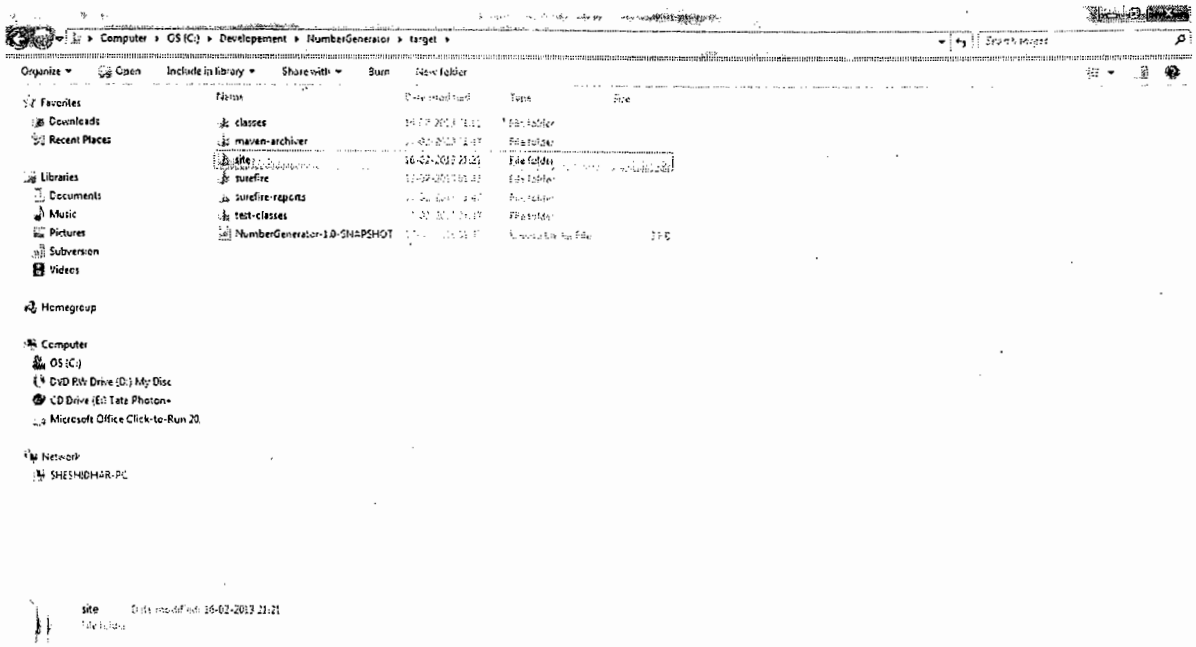
Making a JAR file is straightforward and can be accomplished by executing the following command:

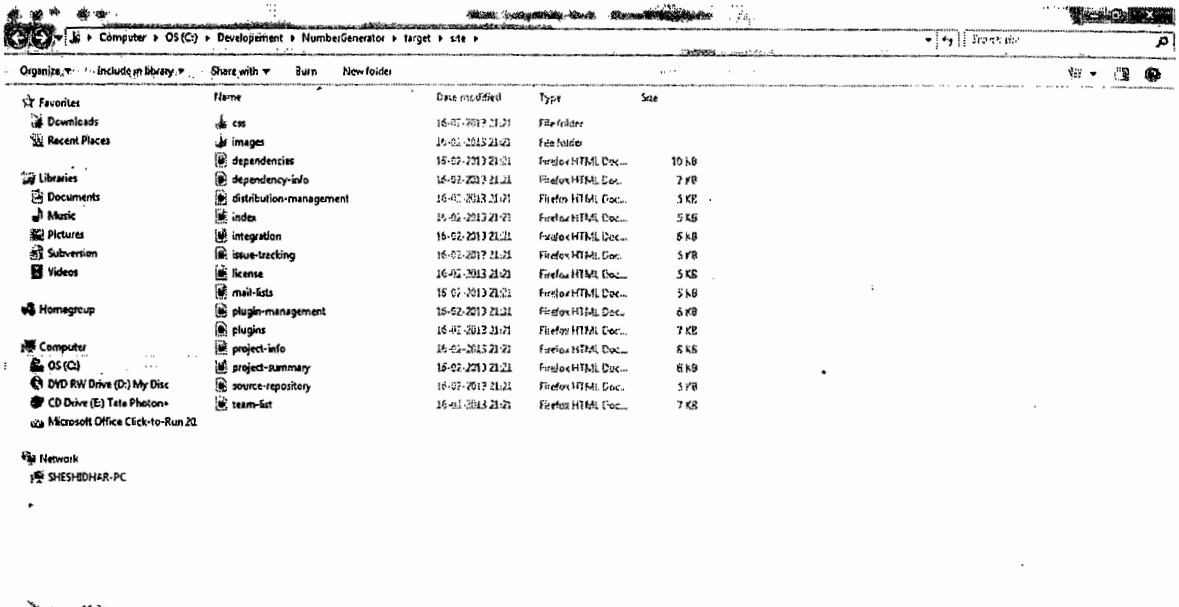
C:\mvnbook\my-app> mvn package

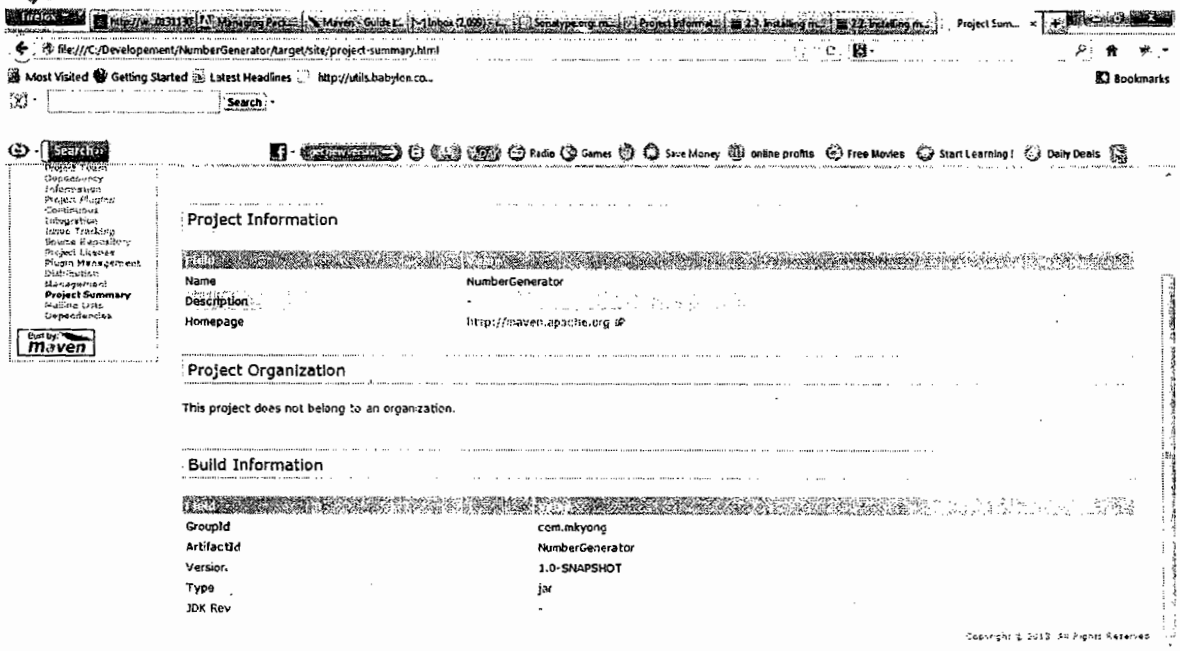


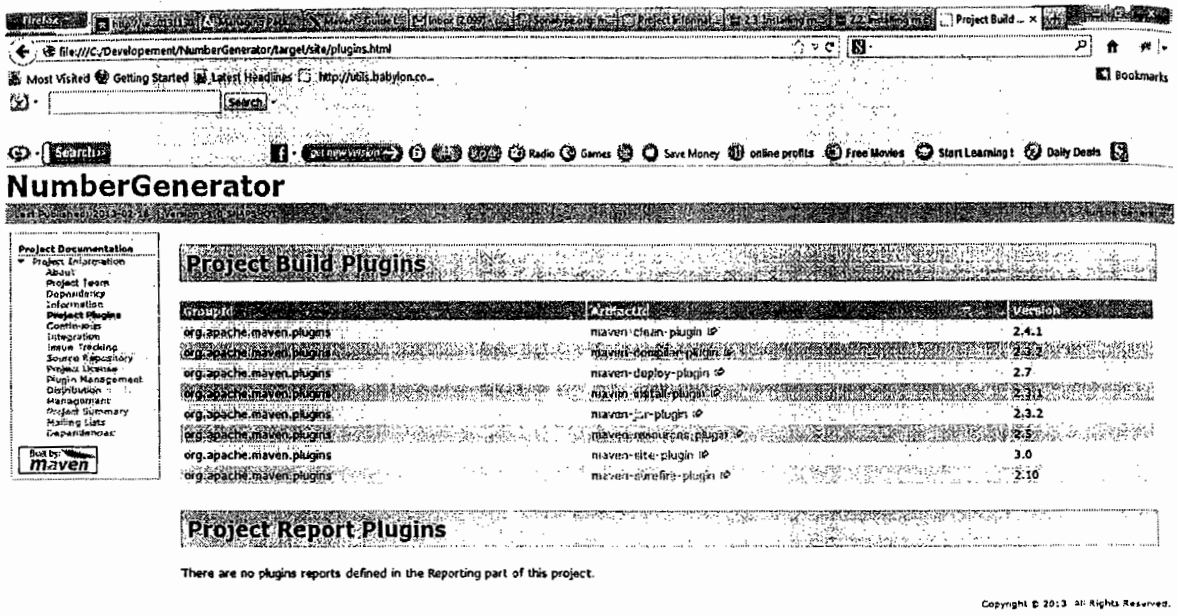
create a basic Web site for your project, simply execute the following command:

C:\mvnbook\my-app> mvn site




Computer ▸ OS (C:) ▸ Developement ▸ NumberGenerator ▸ target ▸ site ▸
Organize
Include in library
Share with
Burn
New folder
Favorites
Downloads
Recent Places
Libraries
Documents
Music
Pictures
Subversion
Videos
Homegroup
Computer
OS (C:)
DVD RW Drive (D:) My Disc
CD Drive (E:) Tata Photon+
Microsoft Office Click-to-Run 20
Network
SHESHIDHAR-PC
Name
Date modified
Type
Size
css
images
dependencies
dependency-info
distribution-management
index
integration
issue-tracking
license
mail-lists
plugin-management
plugins
project-info
project-summary
source-repository
team-list
File folder
Firefox HTML Doc...
16 items


file:///C:/Developement/NumberGenerator/target/site/project-summary.html
Most Visited
Getting Started
Latest Headlines
Bookmarks
Search
Project Information
Name
NumberGenerator
Description
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



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>
```

➢ 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.

Problem Statement :

There are normally multiple people involved in above mentioned deployment process. One team may handles 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>
```

```
<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.

- 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.

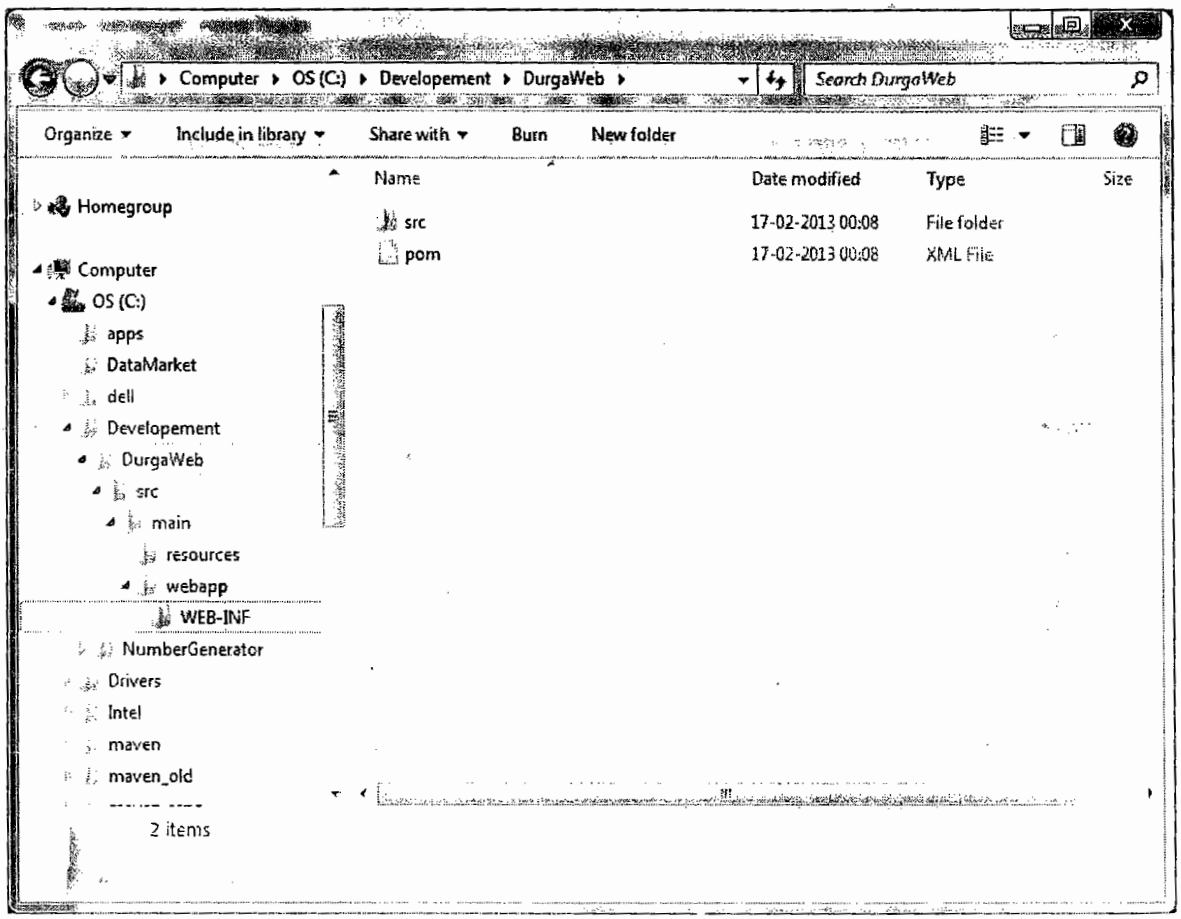
C:\anyfolder>mvn archetype:generate -DgroupId=com.durga.scjp -DartifactId=DurgaWeb -DarchetypeArtifactId=maven-archetype-webapp -DinteractiveMode=false

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
```

```
[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**

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:

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

C:\Development\DurgaWeb>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\DurgaWeb\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\DurgaWeb\target\trucks]
[INFO] Processing war project
[INFO] Copying webapp resources[C:\Development\DurgaWeb\src\main\webapp]
[INFO] Webapp assembled in[77 msecs]
[INFO] Building war: C:\Development\DurgaWeb\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.

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:

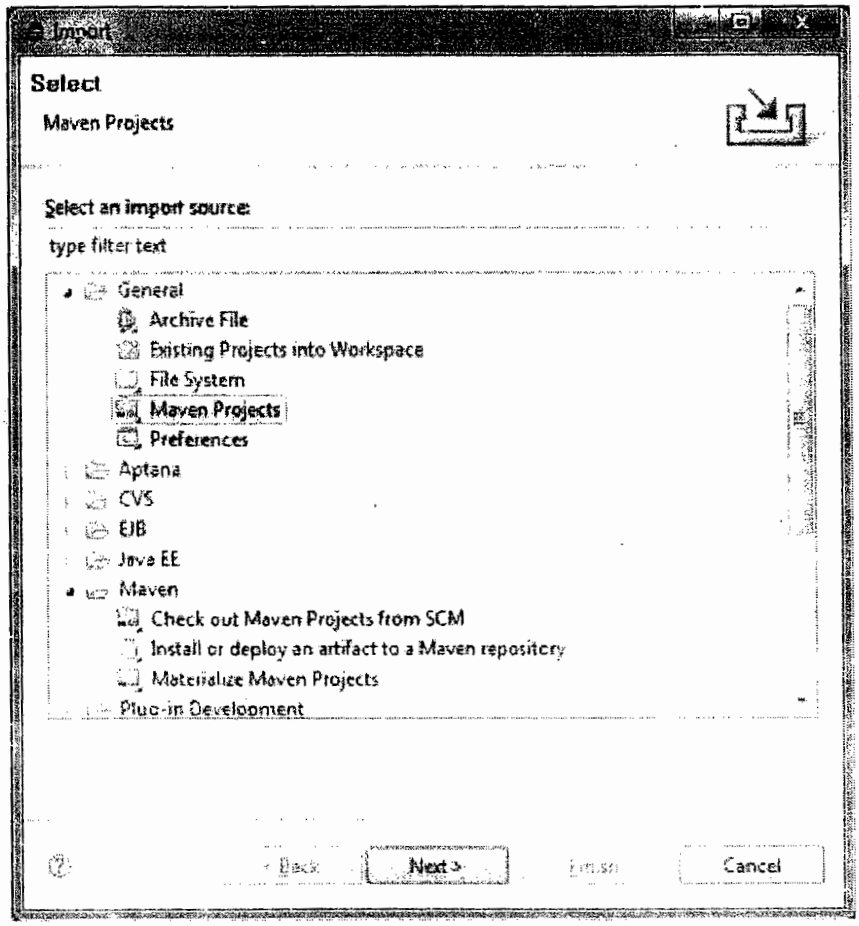
- You can run Maven goals from Eclipse.
- You can view the output of Maven commands inside the Eclipse using its own console.
- You can update maven dependencies with IDE.
- You can Launch Maven builds from within Eclipse.
- It does the dependency management for Eclipse build path based on Maven's pom.xml.
- 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:

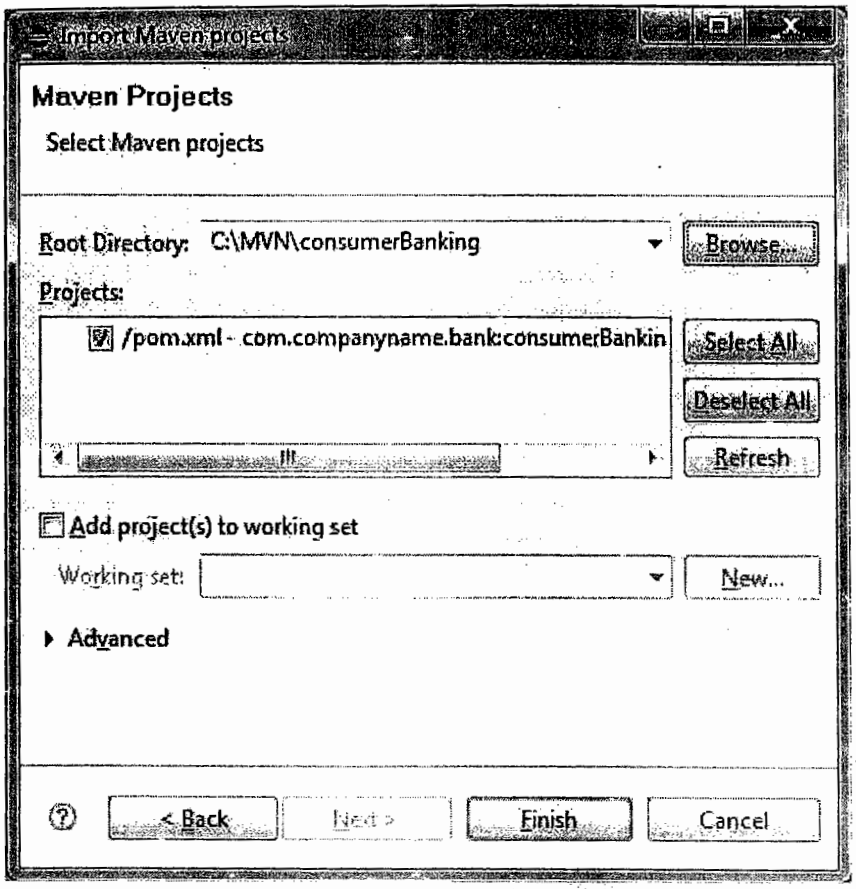
Eclipse	URL
Eclipse 3.5 (Gallileo)	http://www.sonatype.com/books/m2eclipse-book/reference/ch02s03.html
Eclipse 3.6 (Helios)	http://www.sonatype.com/books/m2eclipse-book/reference/install-sect-marketplace.html

Import a maven project in Eclipse:

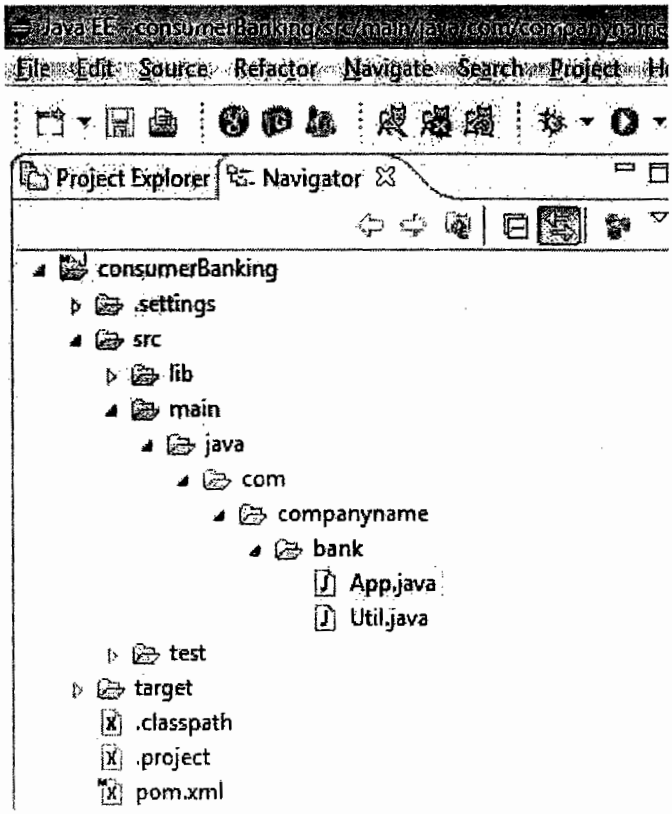
- Open Eclipse.
- Select File > Import > option.
- Select Maven Projects Option. Click on Next Button.



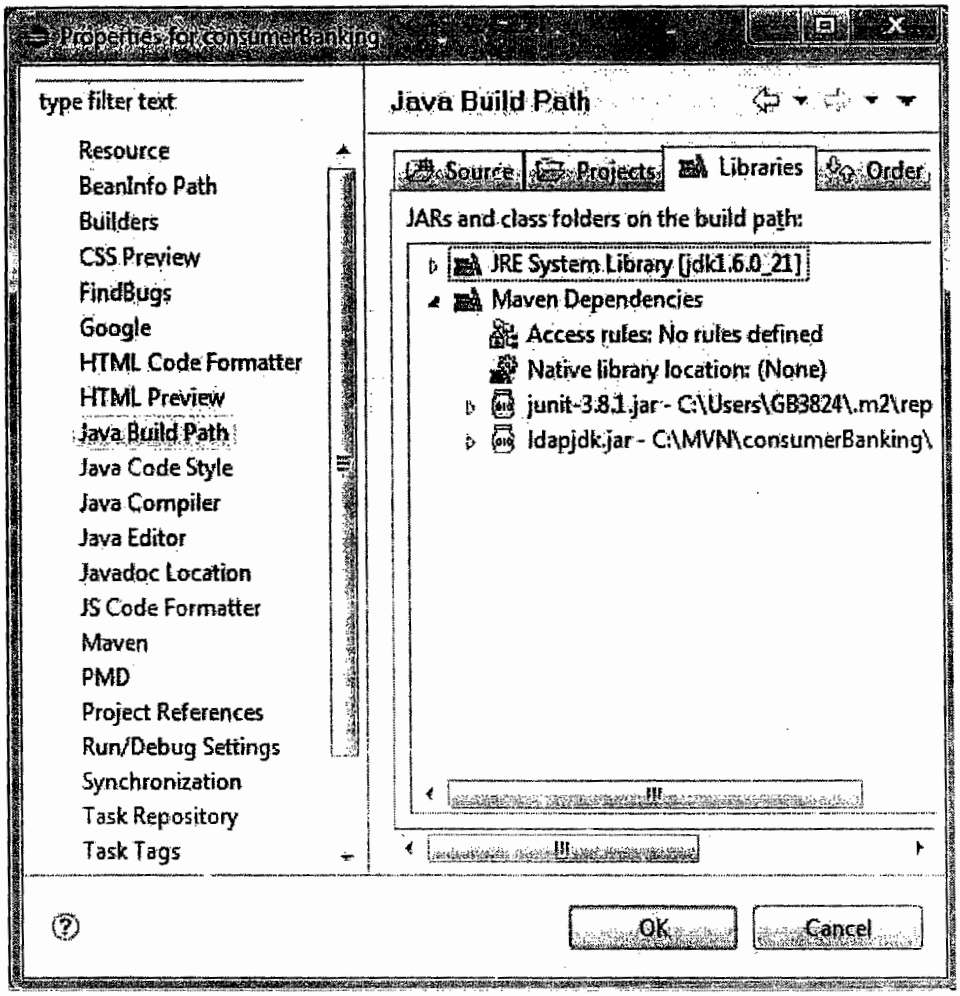
- Select Project location, where a project was created using Maven. We've create a Java Project consumerBanking.See Maven Creating Project to see how to create a project using Maven.
- Click Finish Button.



Now, you can see the maven project in eclipse.



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



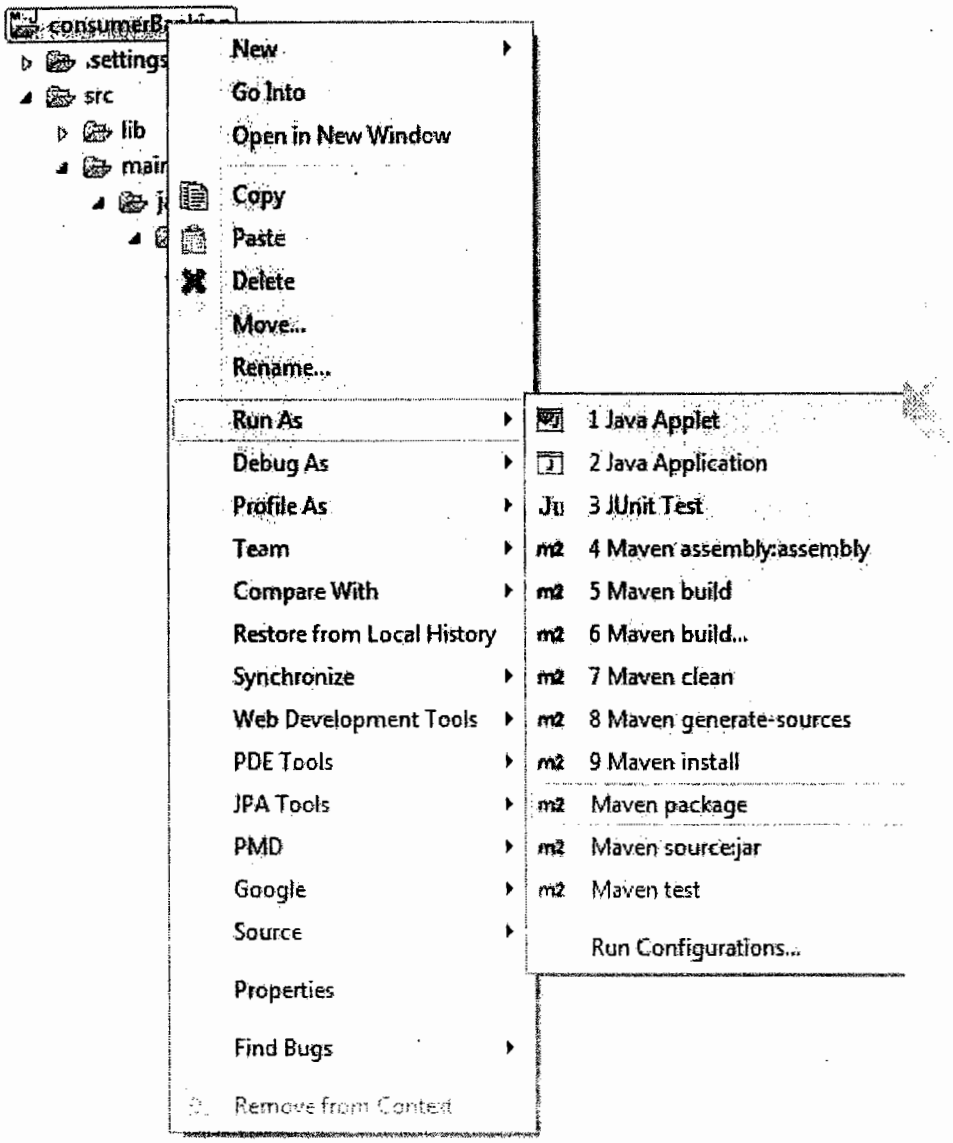
Now, Its time to build this project using maven capability of eclipse.

- 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]

------------------------------------------------------------------------------------------
```

```
[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] ------------------------------------------------------------------
---
```



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

O/P: Hello World!

Maven Programs

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
```

```
[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>
```

```
 <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>
```

```
</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 fie.

```
<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>
```

```
</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 {
```

```
  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>
```

```
<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 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
```

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

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>

        <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">
```

```
        <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 setUserId(int userId) {
```

```
            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.OracleDrive
r</property>
```

```
  <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();
        }

}
```

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)Crate a table CUSTOMER in oracle DB.

CREATE TABLE CUSTOMER

```
( 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>
```

```
    <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

}
```

Data Access Object (DAO) pattern

Craete **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());
```

```
                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) {}
                }
            }
        }
}
```

Spring bean configuration

Create the Spring bean configuration file for customerDAO and datasource.
Crate 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.DriverManagerDataSour
ce">

                <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>
```

Craete 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>
```

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.findByCustomerId(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

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.....

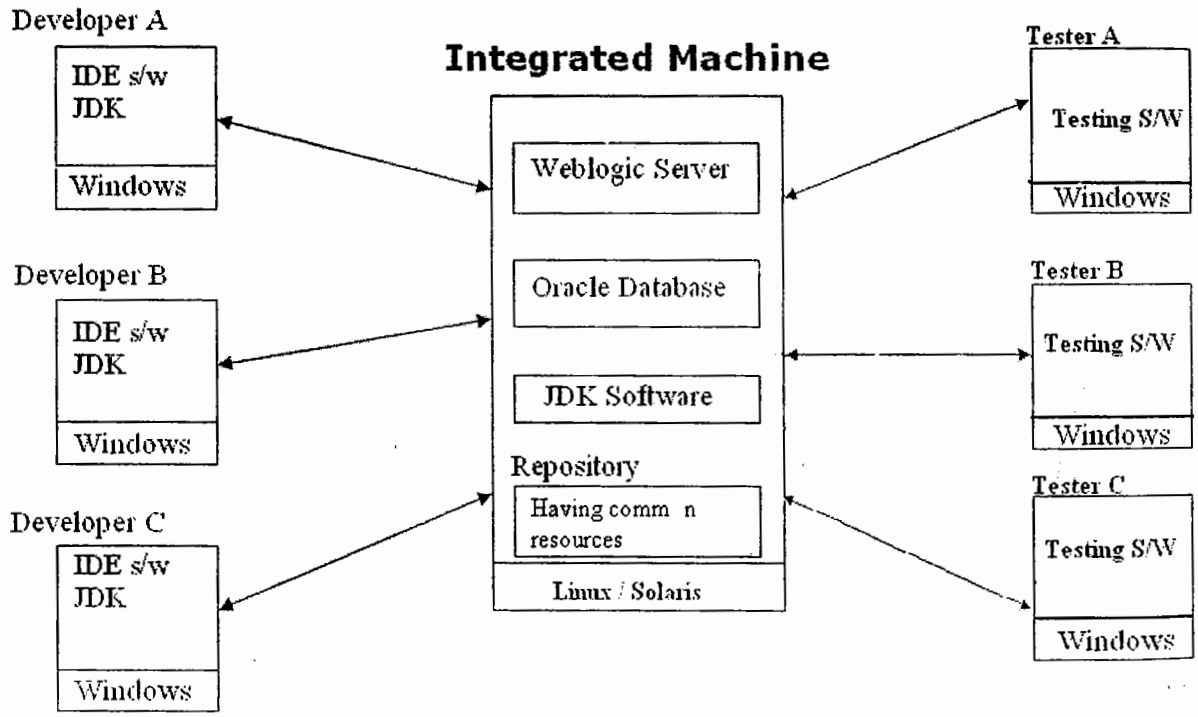
SVN

CONTENT:

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.

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.

- 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 roller back changes).
- ✓ Codes 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)2 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

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.

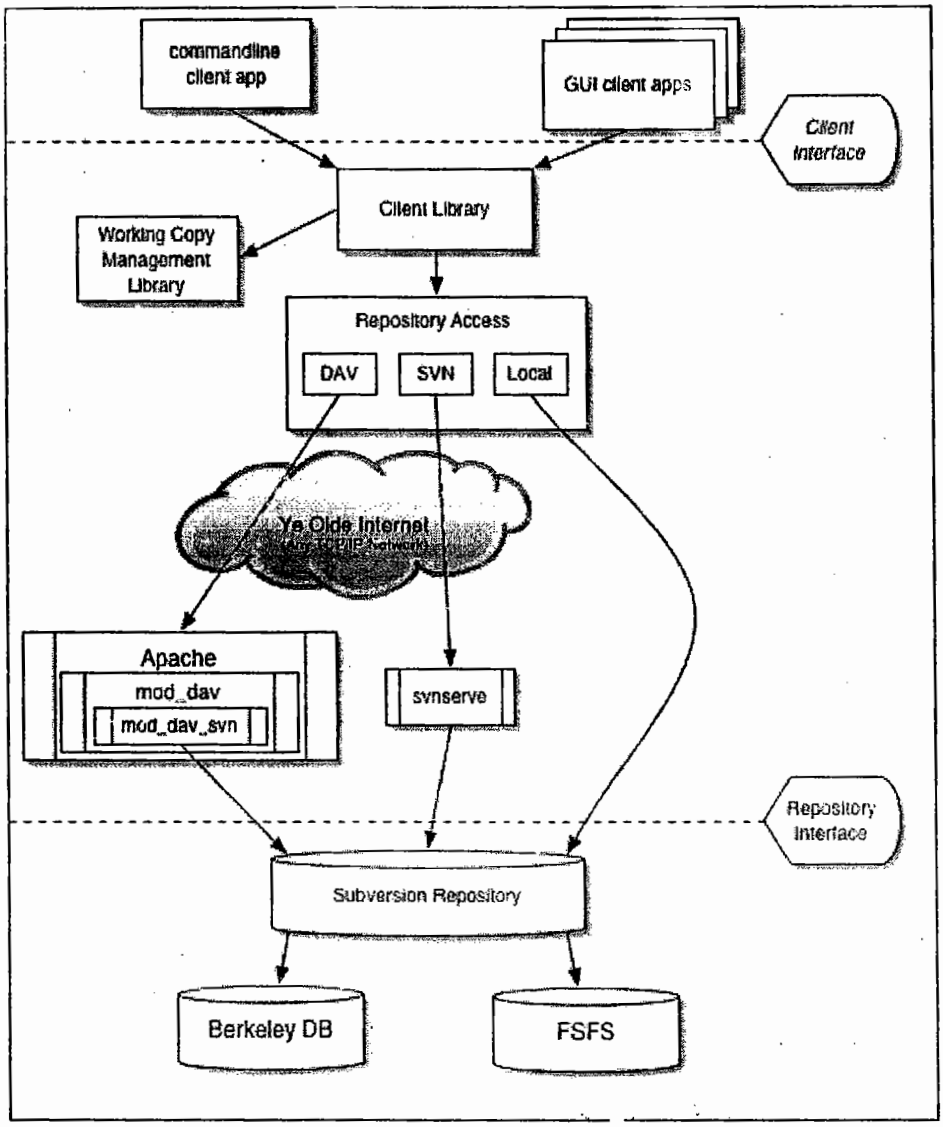
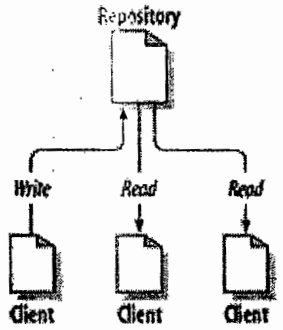


Figure 1.1 Subversion's Architecture

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.



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.

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

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 ate 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.

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.

CVS Vs SVN (Brief Discussion)

As it is known, before getting acquainted with SVN, our company had worked with CVS for long enough. We know CVS from both user and developer points of view. Approximately at the same level we had got acquainted with SVN, and undoubtedly we have an opinion to share with you concerning the question "what system is better?".

It is worth noting, that the work on the creation of the IDE plug-in for SVN had started at numerous user requests. However, our own motivation had also played a considerable role in starting work with SVN, since publications appeared that SVN is a substitute for CVS, which eliminates all its problems and shortcomings. Unfortunately, to our point of view, SVN is not a substitute for CVS and all the more it does not eliminate its shortcomings. Moreover, it even yields to CVS. Figuratively, CVS and SVN can be compared as C ++ and Java. Obviously, both CVS and SVN are more powerful than SourceSafe, as well as C ++ and Java are more powerful than Basic. CVS represents almost all functionalities of a source control system, though not always in a convenient and apparent manner. SVN, patching and expanding some CVS functionalities, simply does not contain some important functions. For example, the creation of tags and branches is dubious, and no means are provided to notify others that you are editing a file. It is similar to what the developers of Java have done: they have decided for you that pointers are not necessary, and there is no need in operator overloading etc.

Thus, as for now SVN cannot be considered a CVS substitute. It is **a different system,** similar to CVS. It has unique functions, which can serve as a reason for its usage. These functions make it more suitable for some development environments, for example for PowerBuilder. Below you can find comparative advantages and disadvantages of these systems. It is assumed that in relation to the remaining items the systems are similar. On the green background "advantages" of a system in relation to its competitor are presented, and "disadvantages" are presented on the pink ground. If you are facing the problem of choice, it is recommended to try both systems, paying special attention to the items below. You may also look to some discussion betwen Subversion developers and Pushok staff.

Note. This comparison cannot be considered as final, since both systems still developed. Last update was made for CVS NT 2.5.01 and SVN 1.2 at July 19 2005.

#	Compared item	CVS	SVN
1	Repository format	CVS is based on RCS files of versions control. Each file connected to CVS is an ordinary	The basis of SVN is a relational database (BerkleyDB) either set of binary files (FS_FS). On one hand,

		file containing some additional information. It is quite natural that the tree of these files repeats the file tree in the local directory. Thus, with CVS you should not be anxious about data loss, and you can easily correct RCS files if necessary.	this settles many problems (for example, concurrent access through the file share) and enables new functionalities (for example, transactions at operations performance). However, on the other hand, data storage now is not transparent, or at least is not available for user interference. That is why the utilities for "curing" and "recovering" of the repository database are provided.
2	Speed	CVS works more slowly.	As a whole, due to some constructive solutions, SVN really works faster than CVS. It transmits less information through the network and supports more operations for offline mode. However, there is the reverse of the medal. Speed increasing is achieved basically at the expense of full backup of all work files on your computer.
3	Tags & Branches (!!! IMPORTANT)	To our point of view, these are implemented properly.	The SVN developers assert with pride that they have got rid of 3 measurements by working with tags and branches. In practice it means that they have substituted both concepts for a capability of copying file(s) or directories within the repository with saving the history of changes. That is, both tag creation and branch creation are substituted for copying within the repository. From the SVN developers' viewpoint, this is very elegant decision, which simplifies one's life. However, we think that there is nothing to be proud of. As for branches, everything is not so bad, now branches are nothing but separate folders in the repository, which earlier have had some interconnection. As for tags,

			everything is much worse. Now you cannot tag a code, this function is simply missing. Certainly, to some extent this is compensated by universal numbering of files in SVN, that is, the whole repository gets the version number, but not each separate file. However, we suppose you will not deny that it is not very convenient to store a four-digit number instead of a symbolic tag.
4	Meta data	CVS allows to store of only files and nothing else.	SVN allows to "attach" to a file any number of all possible named attributes. Excellent functionality, but it is not quite clear what it is necessary for.
5	File types	CVS was initially intended for text data storage. That is why storage of other files (binary, unicode) is not trivial and requires special information, as well as adjustments on either server or client sides.	SVN manipulates all the file types and does not require your instructions.
6	Rollback	CVS allows to rollback any commit in the repository, even if this may require some time (each file should be processed independently)	SVN does not allow rollback of commit. Authors suggest copy good repository state to the end of trunk to overwrite bad commit. However bad commit itself will remain in repository.
7	Transactions	In CVS the support of transactions by the principle "all or nothing" is completely absent. For example, when you checkin several files (transfer them to the server), it is possible that the operation will be completed only for some of these files, and will not be completed for the rest (due to conflicts, for example). As a rule, it is sufficient to correct the situation and to repeat the operation for the remaining files (not for all files). That is, the files	SVN does support transactions by the principle "all or nothing". Probably this is an advantage of the system.

		will be checked in in two steps. No cases of the repository damage due to absence of this functionality were observed.	
8	Availability	Presently CVS is supported everywhere where you might need it.	SVN not yet so widly used as the result there are places where it support still not implemented.
9	Internal architecture and code.	CVS is very old system. Initially CVS was written as bunch of scripts around RCS executable. Later this packaget into single executable. But internal structure of code is poor and have lots of historical fixes. Till now there was several attempts to rewrite CVS from scratch, but as it is known without success. We personally tried to extract client code to make better integration between plug-in and CVS, but without success. Right now we not think that CVS may grows too much in its functionality.	Subversion authors really spent some time on internal SVN architecture. Still not know how good are some decisions they make. But one is clear, the code is well expandable, and future imp

UBER SVN Installation:

SVN softwares comes in different flavor like

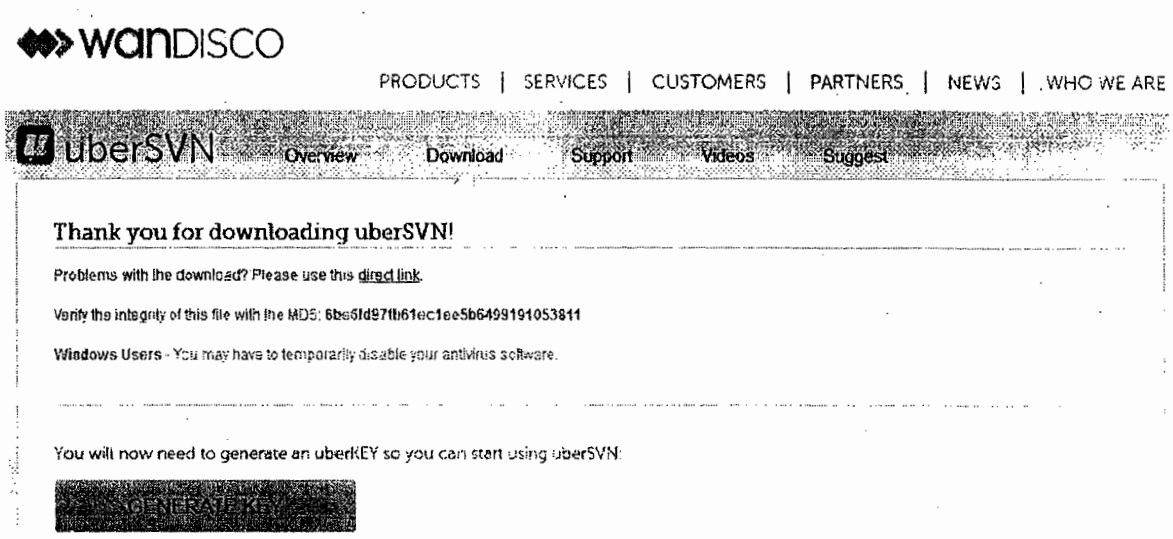
- CollabNet (supported and certified by CollabNet; *requires registration*)
- SlikSVN (32- and 64-bit client MSI; maintained by Bert Huijben, SharpSvn project)
- VisualSVN (client and server; supported and maintained by VisualSVN)
- WANdisco (32- and 64-bit client and server; supported and certified by WANdisco)
- Win32Svn (32-bit client, server and bindings, MSI and ZIPs; maintained by David Darj)
- uberSVN(32 and 64 bit Server)

We also get purely svn client like Tortoise SVN.
We are going to use uberSVN and Tortoise SVN.

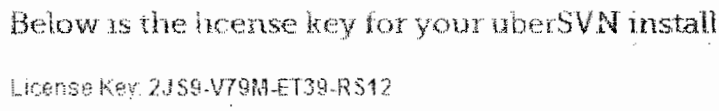
DOWNLOADING UberSVN

download the server using this link.

http://www.wandisco.com/ubersvn/download#windows



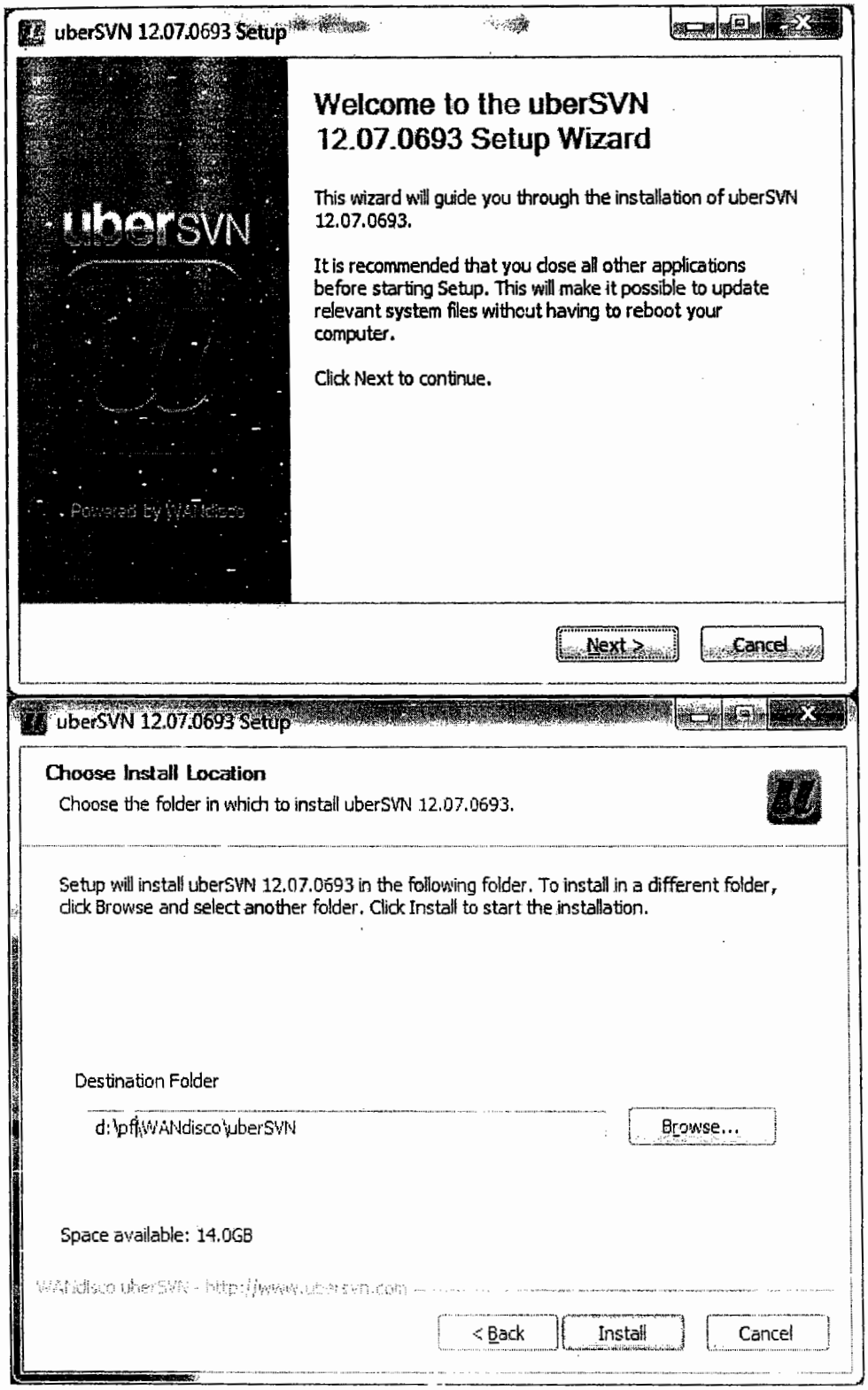
After download is complete,You need to generate a registration key(dont wory its free).
Click on Generate Key.
If you have not registered,A new window will be opened and You will be asked to register.Register using your valid email id,and then u will get an activation mail on your email.click on the activation mail,and your account will be active.
Again click on generate Key,button

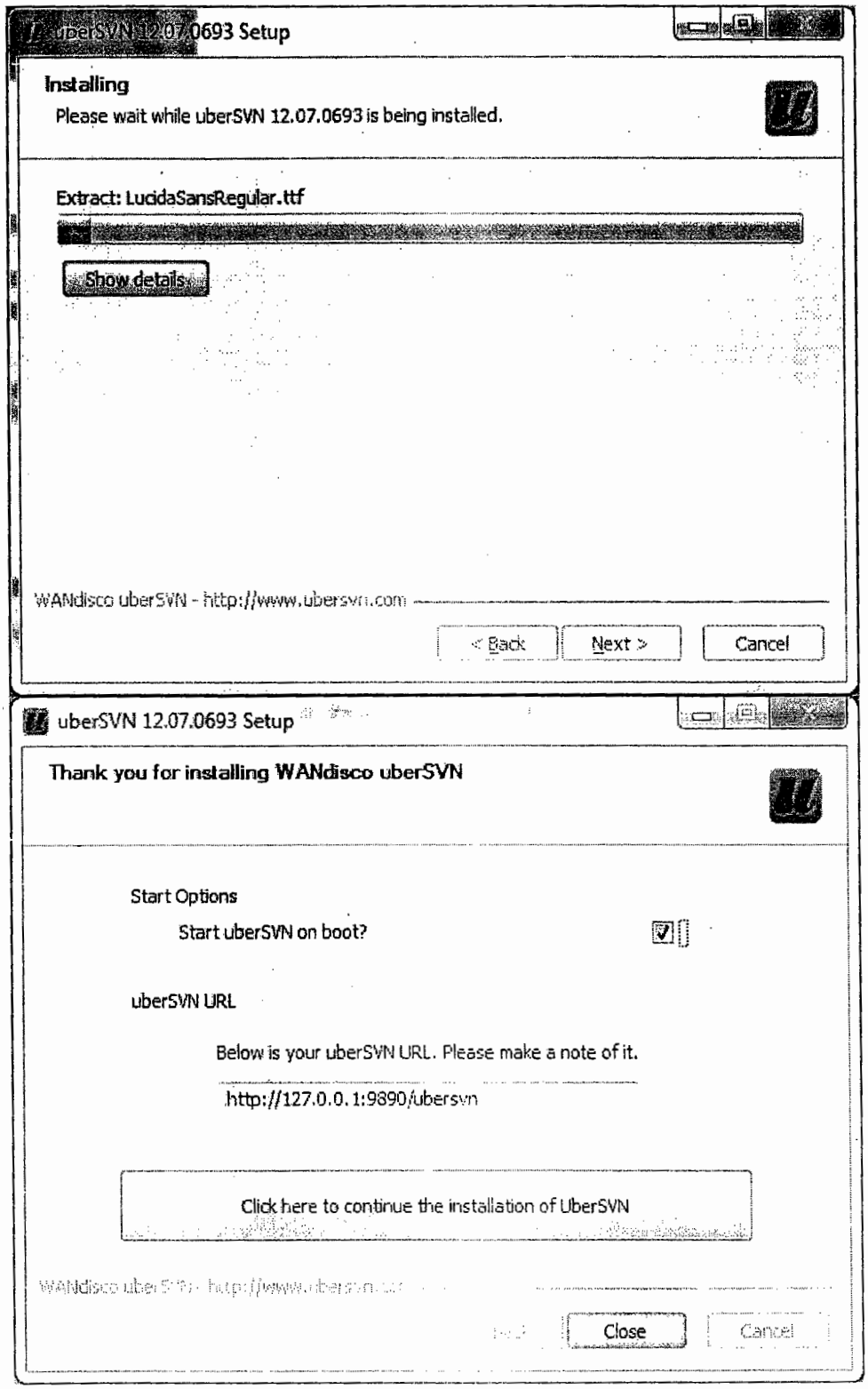


Downloading Tortoise SVN,
Use the link to download tortoise svn

http://tortoisesvn.net/downloads.html

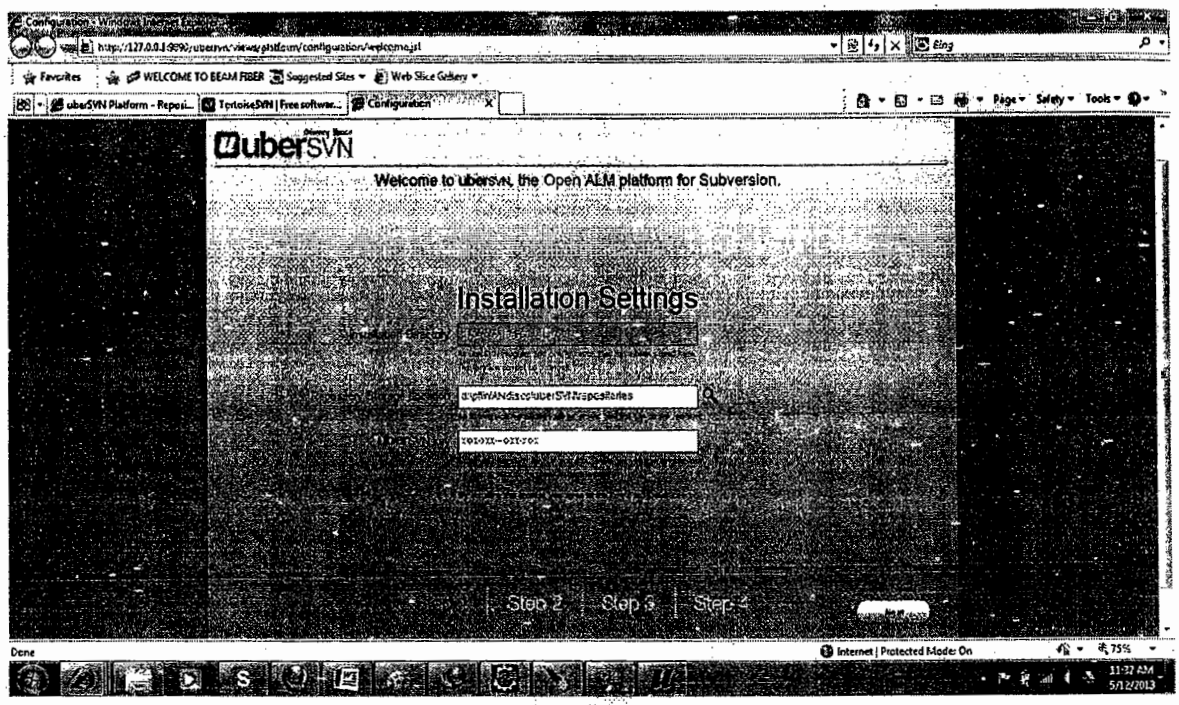
Installing UBER SVN


uberSVN 12.07.0693 Setup
uberSVN
Powered by WANdisco
Welcome to the uberSVN 12.07.0693 Setup Wizard
This wizard will guide you through the installation of uberSVN 12.07.0693.
It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.
Click Next to continue.
Next >
Cancel
uberSVN 12.07.0693 Setup
Choose Install Location
Choose the folder in which to install uberSVN 12.07.0693.
Setup will install uberSVN 12.07.0693 in the following folder. To install in a different folder, click Browse and select another folder. Click Install to start the installation.
Destination Folder
d:\pf\WANdisco\uberSVN
Browse...
Space available: 14.0GB
WANdisco uberSVN - http://www.uberSVN.com
< Back
Install
Cancel



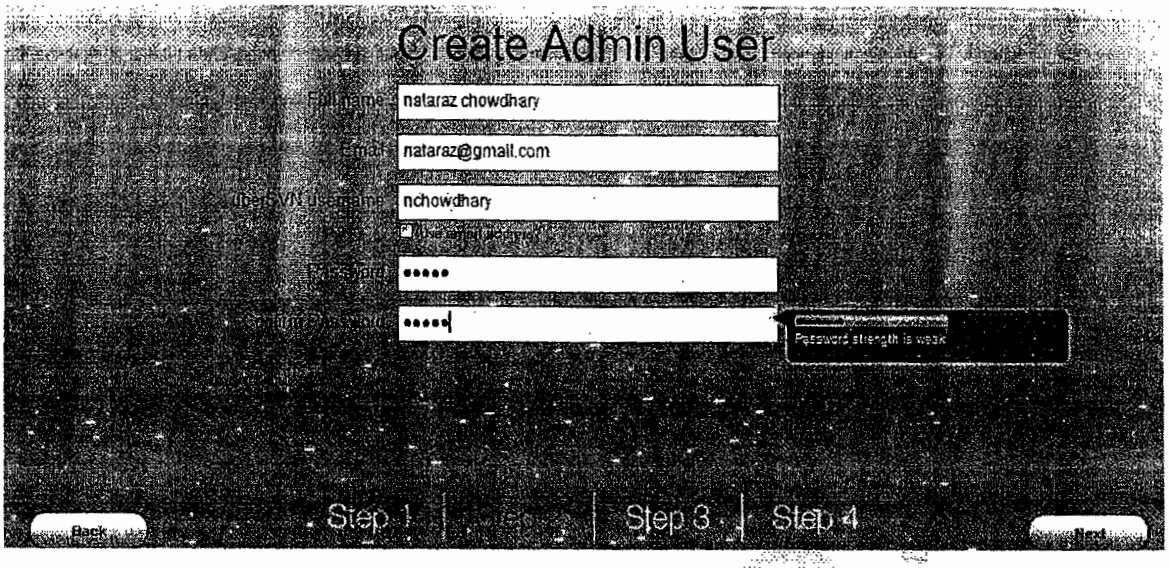
Click on click here to continue the installation of UberSVN button,You will get the homepage in browser window.

Choose your repository storage location and submit the uberSVN Key,which we had generated previously



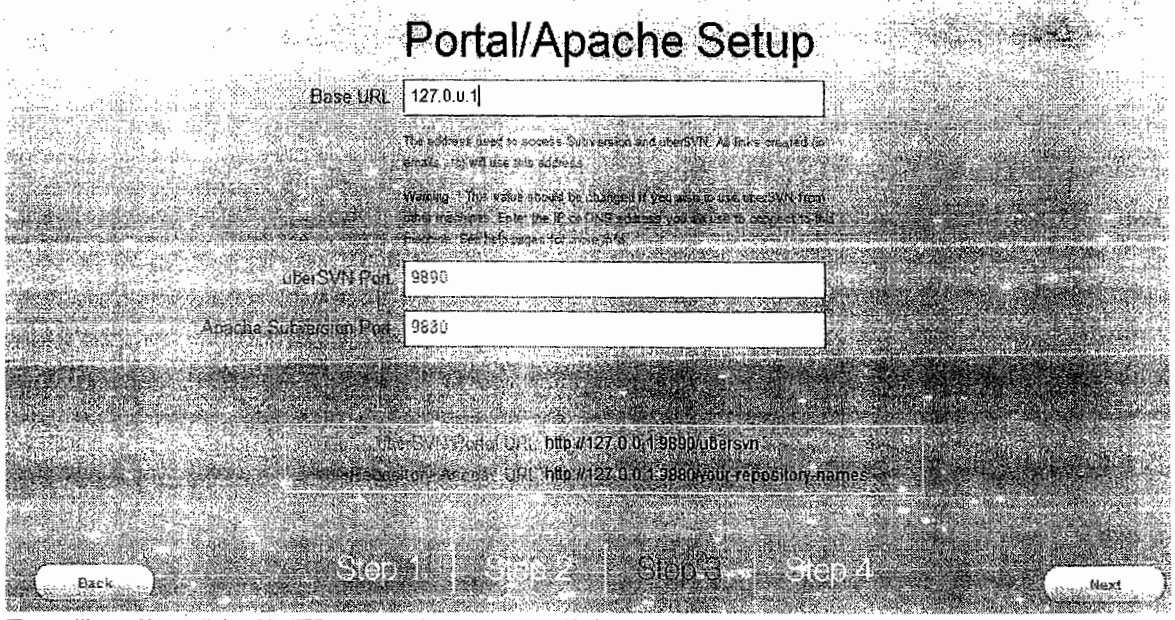
Click on next button

Create Admin User

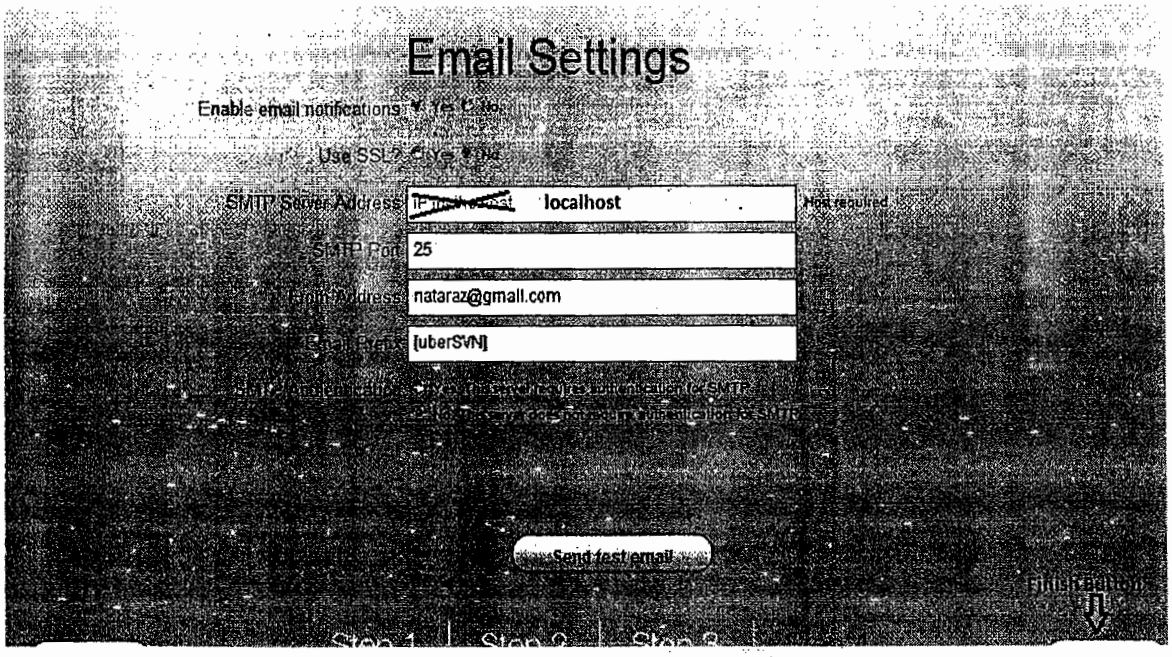


Change Port Number,If needed,and click next

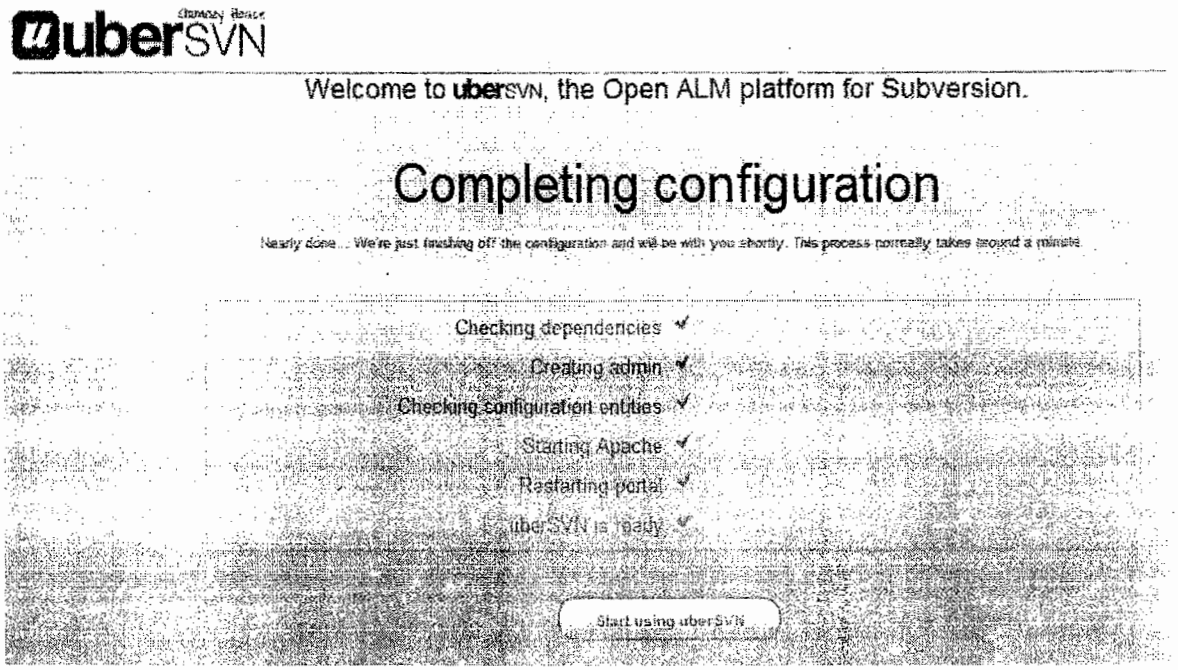
Welcome to uberSVN, the Open ALM platform for Subversion.



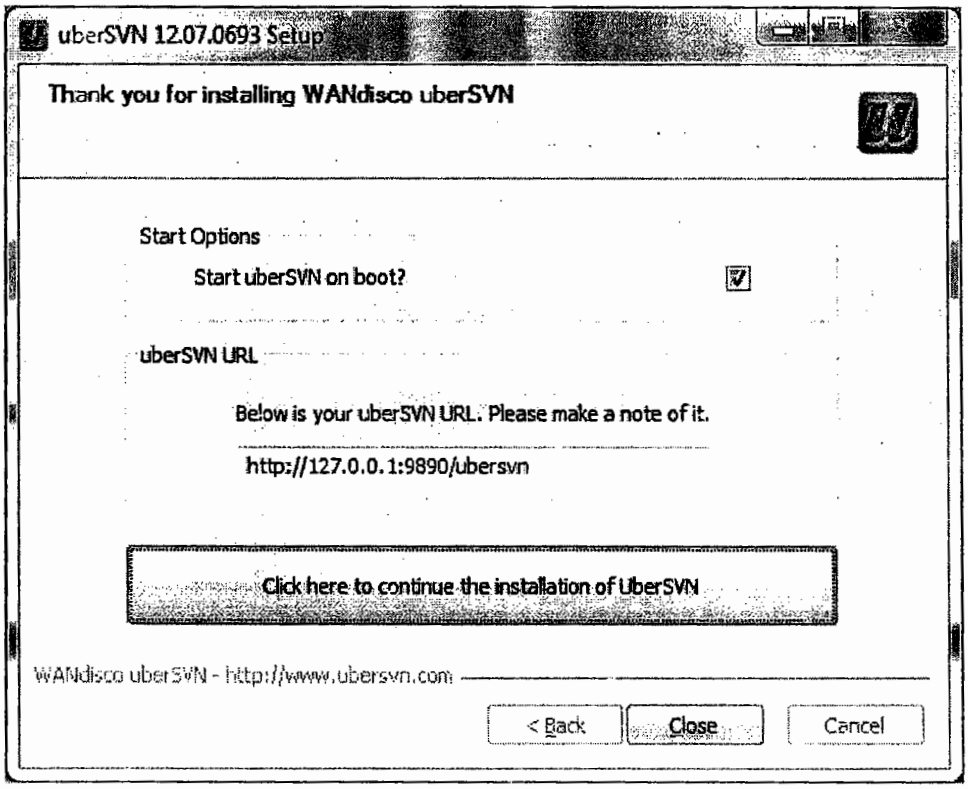
Type "**localhost**" in SMTP Server Name,and click on Finish



You will get Complete Configuration message.

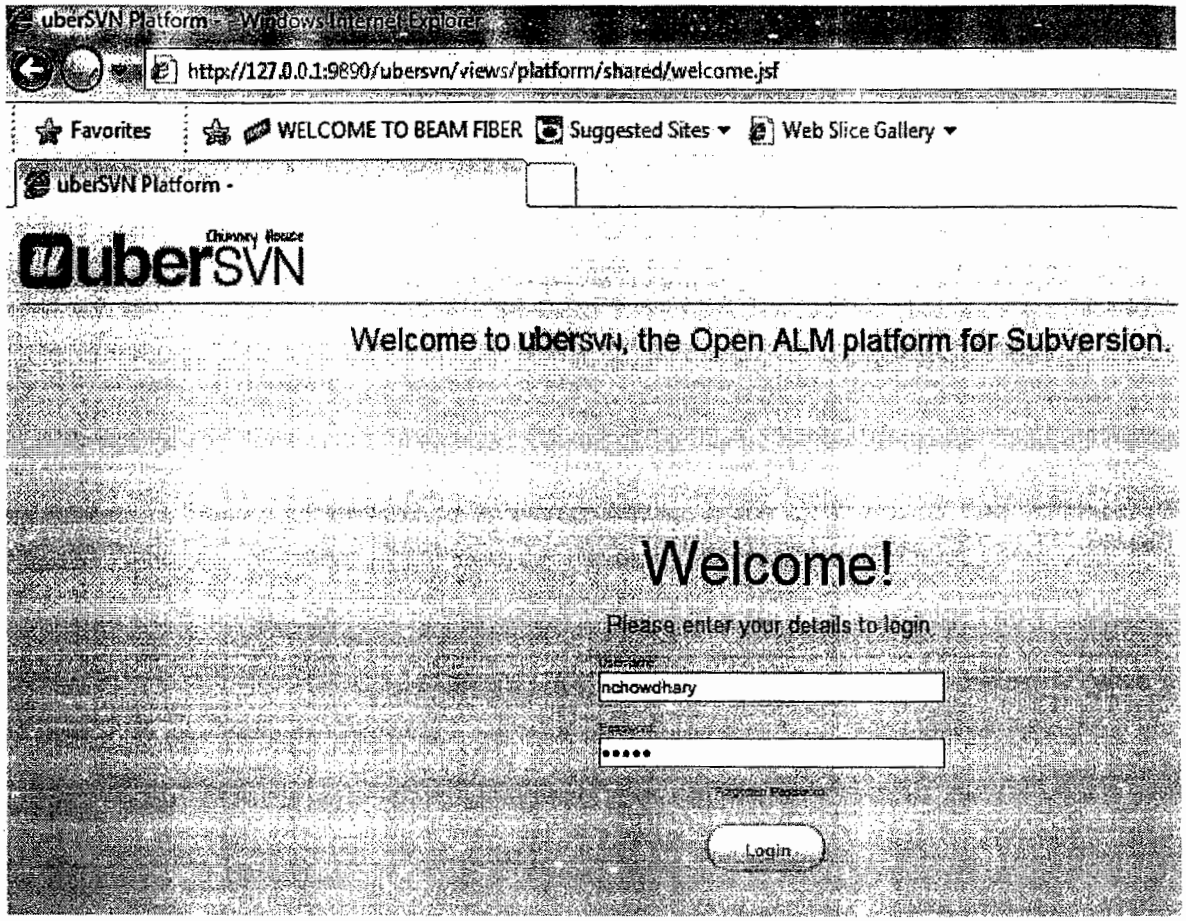


Click on Start Using uberSVN,You will now get the login page for admin
username:nchowdhary
password:12345
You can now click close button of Previous window

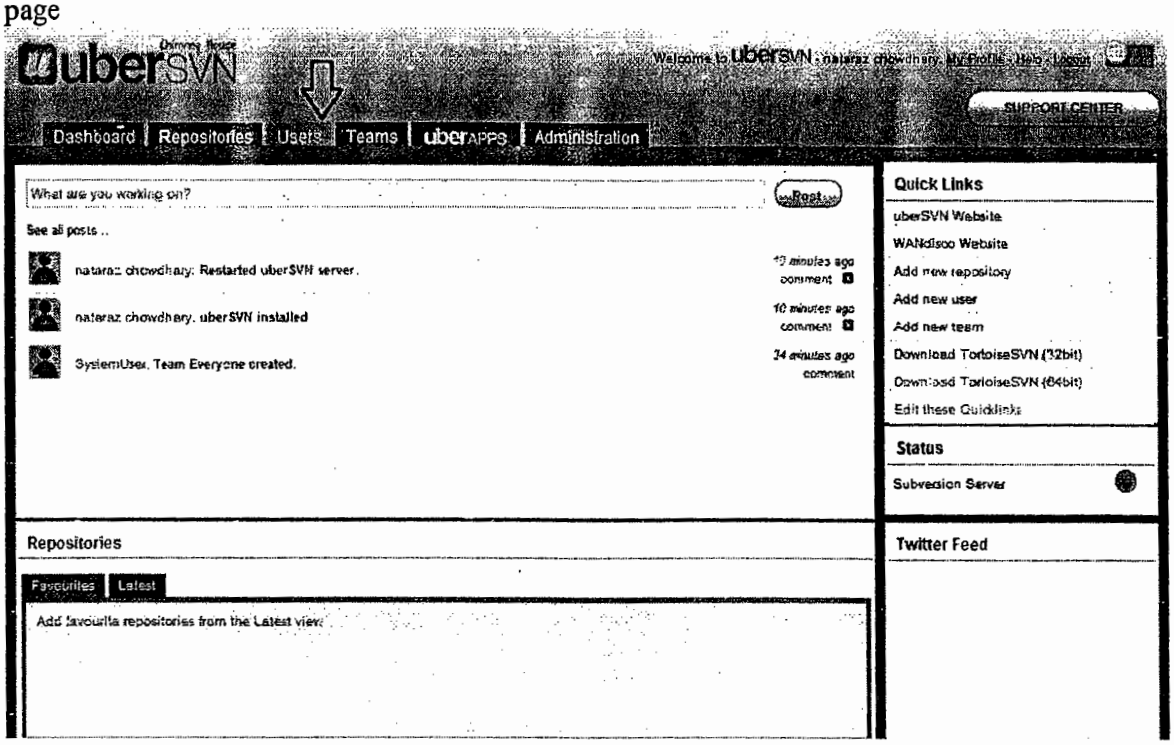


Login into SVN

Use the above credential to login to the application

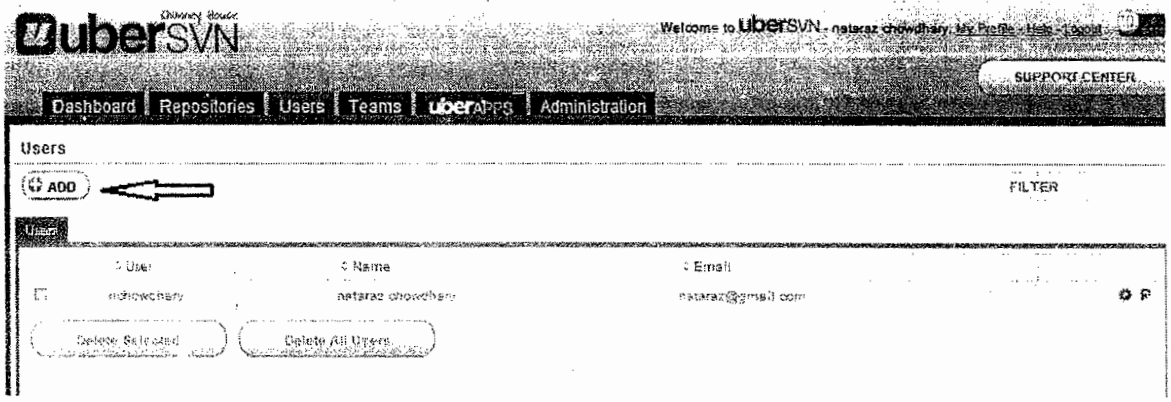

uberSVN Platform - Windows Internet Explorer
http://127.0.0.1:9890/ubersvn/views/platform/shared/welcome.jsf
Favorites
WELCOME TO BEAM FIBER
Suggested Sites
Web Slice Gallery
uberSVN Platform -
uberSVN
Welcome to ubersvn, the Open ALM platform for Subversion.
Welcome!
Please enter your details to login
nchowdhary
Login

When we click Login,we will get the below page

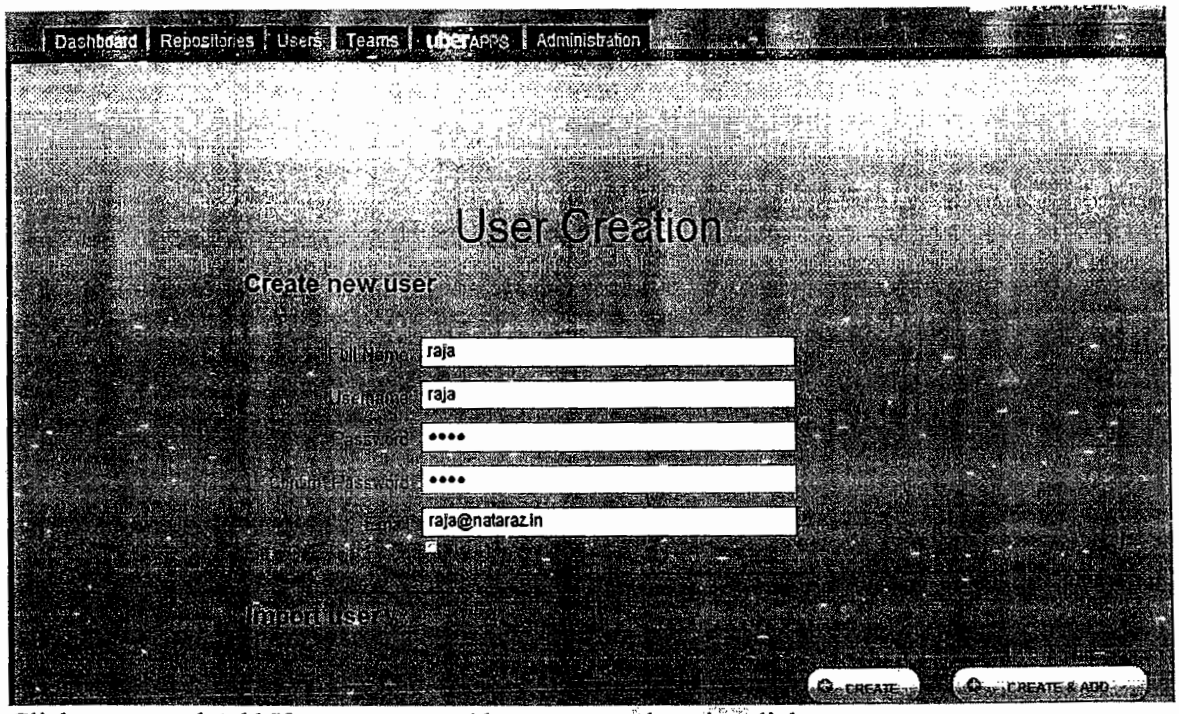


Lets add a team,which contains few members,and those members are allowed to access the svn.

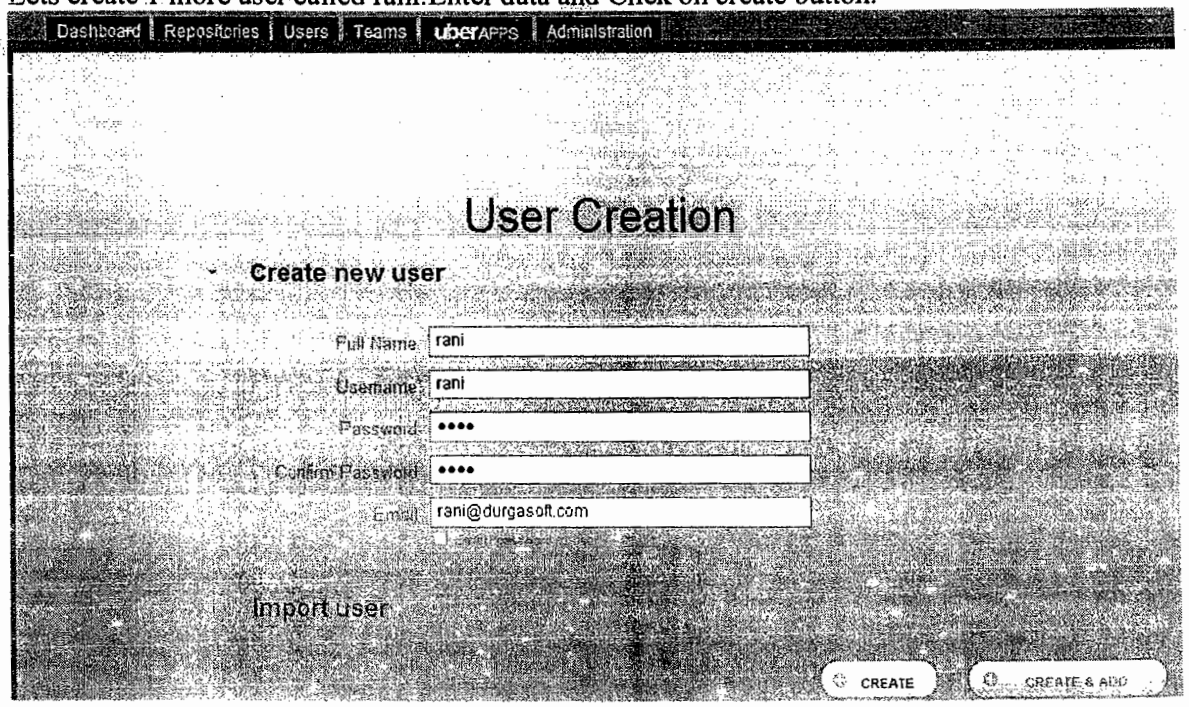
Create 2 user,Click on user tab,click Add



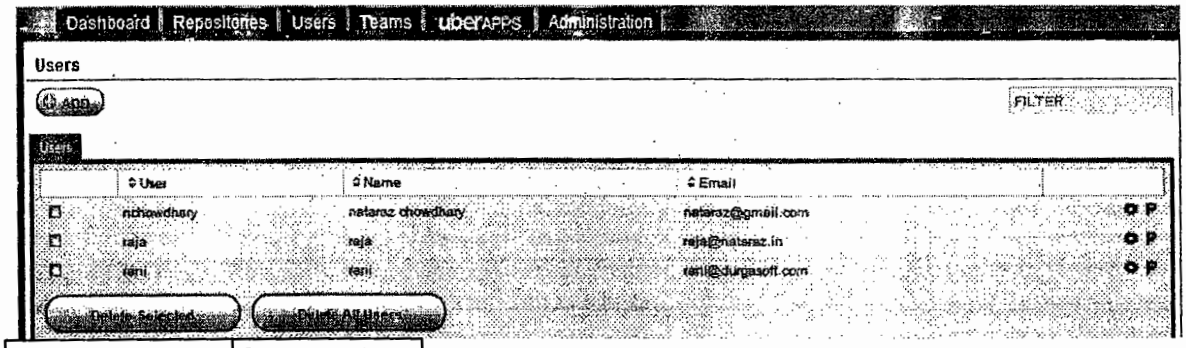
Add the User Details



Click create and Add,If you want to add more user,otherwise click create.
Lets create 1 more user called rani.Enter data and Click on create button.

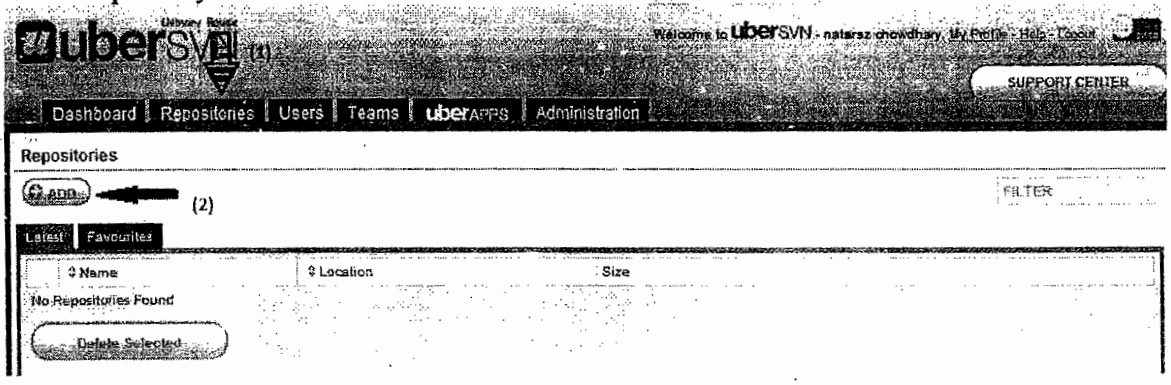


By the end of this,we have created 3 users.

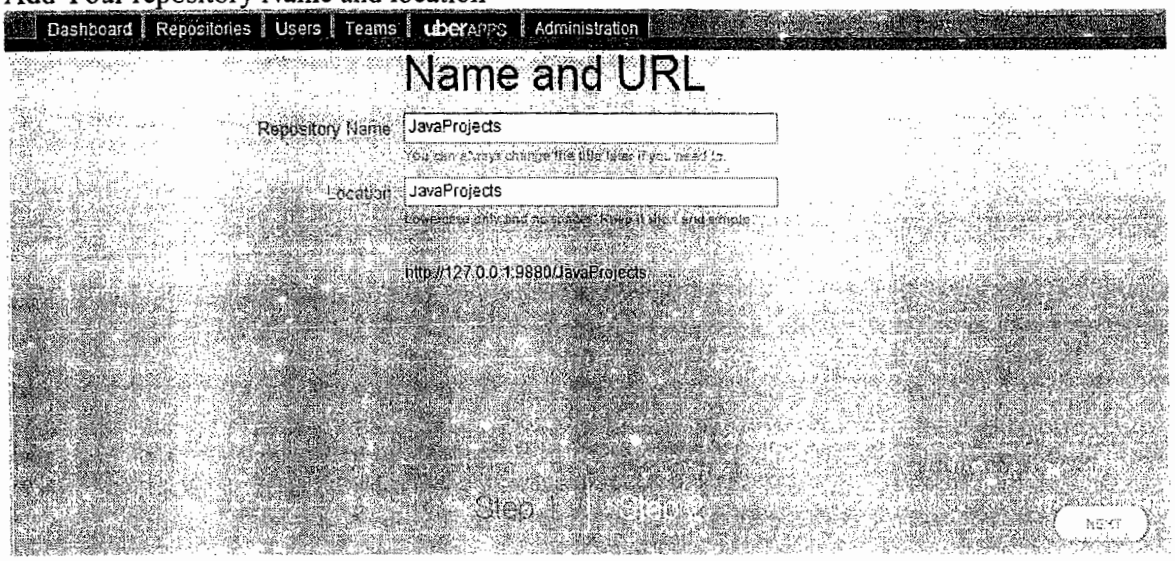


User ID	Password
nchowdhary	12345
Raja	Raja
rani	Rani

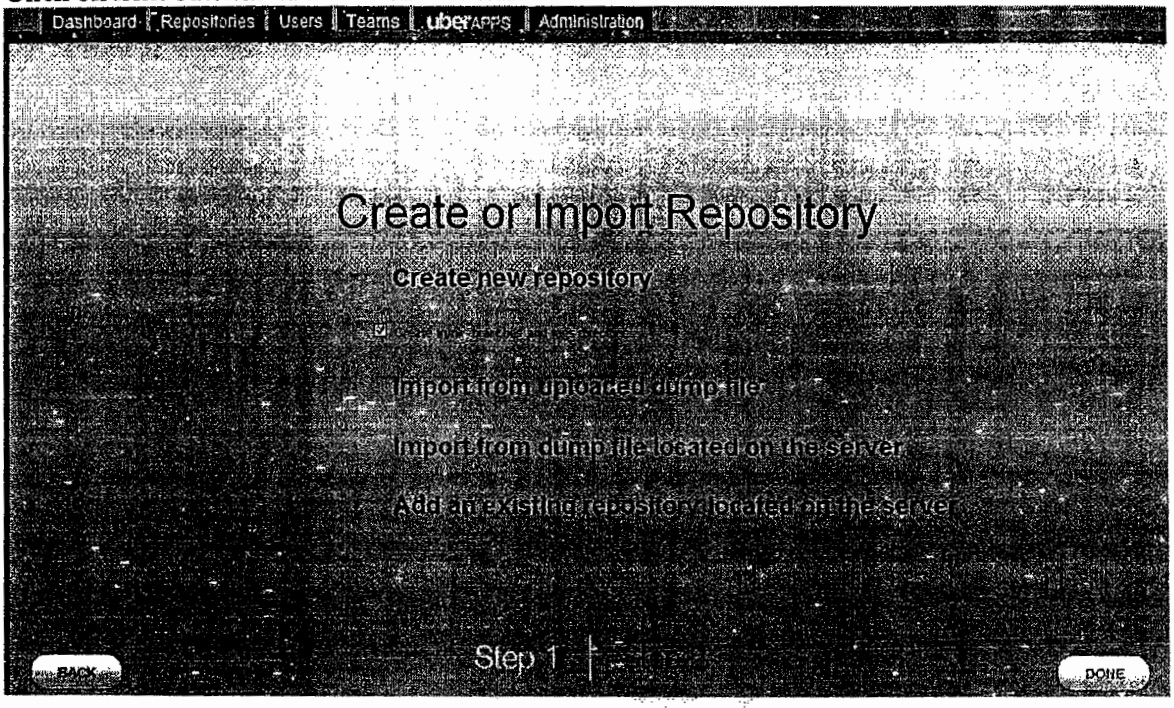
For Every Big project,We have a repository,where we store all our projects.
Lets create a Repository,
Go to Repository tab and click add button.



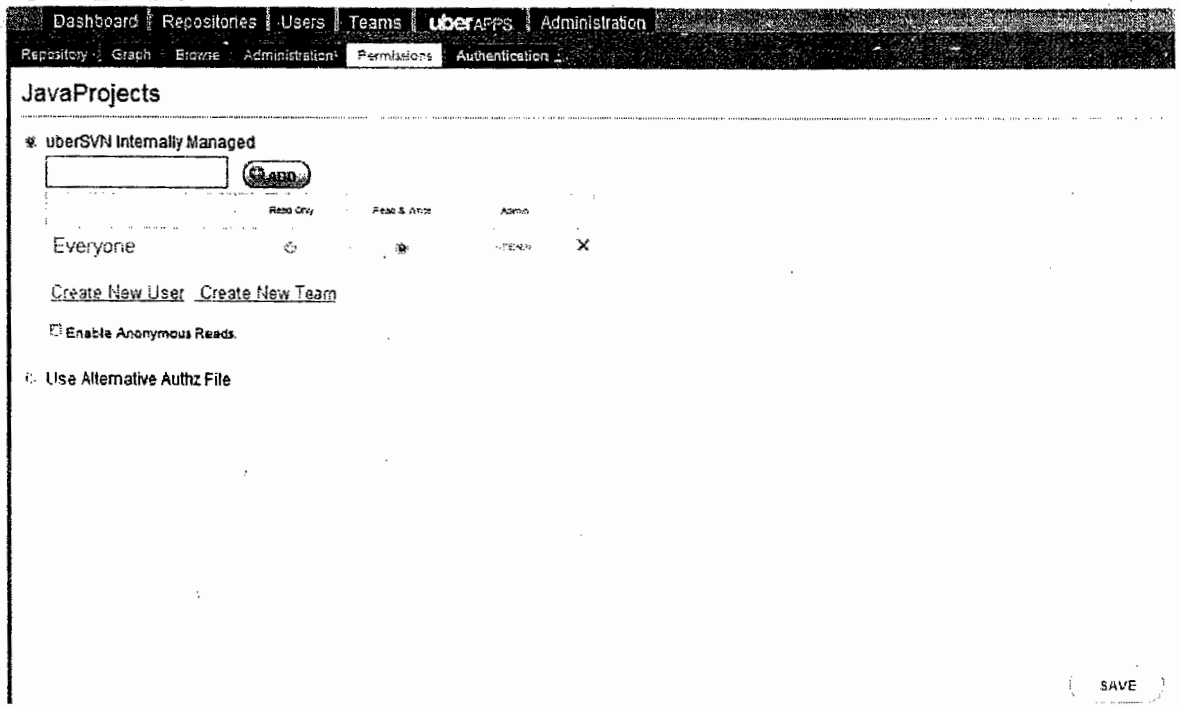
Add Your repository Name and location



Click on next button.and then Done button



Click on "Save"



So we have created Users(raja,rani) and Repositories(Java Projects).Now We need to create a team,linking users and repositories.

Creating Team

Go to Teams tab,and click on Add button