

**LAKIREDDY BALI REDDY COLLEGE OF ENGINEERING  
(AUTONOMOUS)**

**Continuous Integration and Continuous Delivery using  
Devops  
IV Year I Semester (R20)**



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**DevOps-Continuous Integration and Continuous Delivery**

### **Vision of the Department**

The Computer Science & Engineering aims at providing continuously stimulating educational environment to its students for attaining their professional goals and meet the global challenges.

### **Mission of the Department**

- **DM1:** To develop a strong theoretical and practical background across the computer science discipline with an emphasis on problem solving.
- **DM2:** To inculcate professional behaviour with strong ethical values, leadership qualities, innovative thinking and analytical abilities into the student.
- **DM3:** Expose the students to cutting edge technologies which enhance their employability and knowledge.
- **DM4:** Facilitate the faculty to keep track of latest developments in their research areas and encourage the faculty to foster the healthy interaction with industry.

### **Program Educational Objectives (PEOs)**

- **PEO1:** Pursue higher education, entrepreneurship and research to compete at global level.
- **PEO2:** Design and develop products innovatively in computer science and engineering and in other allied fields.
- **PEO3:** Function effectively as individuals and as members of a team in the conduct of interdisciplinary projects; and even at all the levels with ethics and necessary attitude.
- **PEO4:** Serve ever-changing needs of society with a pragmatic perception.

### **PROGRAMME OUTCOMES (POs):**

<b>PO 1</b>	<b>Engineering knowledge:</b> Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
<b>PO 2</b>	<b>Problem analysis:</b> Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
<b>PO 3</b>	<b>Design/development of solutions:</b> Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
<b>PO 4</b>	<b>Conduct investigations of complex problems:</b> Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
<b>PO 5</b>	<b>Modern tool usage:</b> Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
<b>PO 6</b>	<b>The engineer and society:</b> Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

## **DevOps-Continuous Integration and Continuous Delivery**

<b>PO 7</b>	<b>Environment and sustainability:</b> Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
<b>PO 8</b>	<b>Ethics:</b> Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
<b>PO 9</b>	<b>Individual and team work:</b> Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
<b>PO 10</b>	<b>Communication:</b> Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
<b>PO 11</b>	<b>Project management and finance:</b> Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
<b>PO 12</b>	<b>Life-long learning:</b> Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change

**PROGRAMME SPECIFIC OUTCOMES (PSOs):**

<b>PSO 1</b>	The ability to apply Software Engineering practices and strategies in software project development using open-source programming environment for the success of organization.
<b>PSO 2</b>	The ability to design and develop computer programs in networking, web applications and IoT as per the society needs.
<b>PSO 3</b>	To inculcate an ability to analyze, design and implement database applications.

# DevOps-CICD

(Continuous Integration and Continuous Delivery)

---

## CONTENTS

### Context...

Exercise 1: Testing lab setup

Exercise 2: Git operations

Exercise 3: Creating a project in SonarQube

Exercise 4: Using Sonarqube with Sonar-Runner

Exercise 5: Creating a local repository in Artifactory

Build Automation: Maven

Exercise 6: Build automation using Maven

Continuous Integration: Jenkins

Exercise 7: Jenkins Installation & System Configuration

Exercise 8: Download the plugins in Jenkins

**DevOps-Continuous Integration and Continuous Delivery**

Exercise 9: Creating Central CI pipeline

Exercise 10: Copying and Moving Jobs

Exercise 11: Creating pipeline view in Jenkins

Exercise 12: Configuring Gating Conditions

### **Additional Exercises**

1. Adding custom rules to SonarQube
2. Static program analysis using SonarLint
3. Binding SonarQube rules to SonarLint

## **Context**

This document contains exercises and activities that would be done during the lab practice of DevOps. This would provide hands on experience on the concepts and help the participants create pipelines using open source tool stack.

## **Application Used:**

A simple web application using Maven.

## **Tools that would be used:**

- Eclipse – Integrated development environment
- JUnit – Unit testing of code
- Jenkins – Continuous integration server
- Git- Source code management
- Jenkins – Build Automation
- SonarQube- Source code quality management
- JaCoCo – Code coverage

## **NOTE:**

- For performing the exercises below, participants must have the setup of the tools mentioned.
- The screenshots provided are illustrative, based on your system setup, the contents may change.

## **Exercise 1: Testing lab set up**

**Step 1:** Open the System Environment Variables by right-click on This PC -> Properties -> Advanced System Settings -> Environment Variables..

Add the following as System Variables if not added already:

- o JAVA\_HOME = path to jdk folder (C:\Program Files\Java)
- o M2\_HOME = path to maven folder (C:\Program Files\Maven)
- o PATH = add “%JAVA\_HOME%/bin; %M2\_HOME%/bin; <path to sonar-runner>\sonar-runner-2.4\bin;” to the existing path variables.

**Step 2:**Start all the installed tools by executing the appropriate batch file.

**Step 3:**you can ensure all are working by accessing tools user interface with belowmentioned URL's and credentials.

1. <http://localhost:8080> – – SonarQube [admin/admin]
2. <http://localhost:8080> – Tomcat [tomcat/s3cret]
3. <http://localhost:8064/jenkins> - Jenkin


**Summary:** You have tested lab set up for the forthcoming exercises for Jenkins.

## **Exercise 2: Git operations**

Objective: Perform the basic operations on Git repositories using EGit (Eclipseplugin).

### **Pulling code from Git repository**

**Step 1:**Go to Eclipse-> file->import->git->Projects from git->clone uri ->enter central repository details -> use credentials.

**Source Git Repository** 

Enter the location of the source repository.

Location

URI:

Host:

Repository path:

Connection

Protocol:

Port:

Authentication

User:

Password:

☐ Store in Secure Store

?

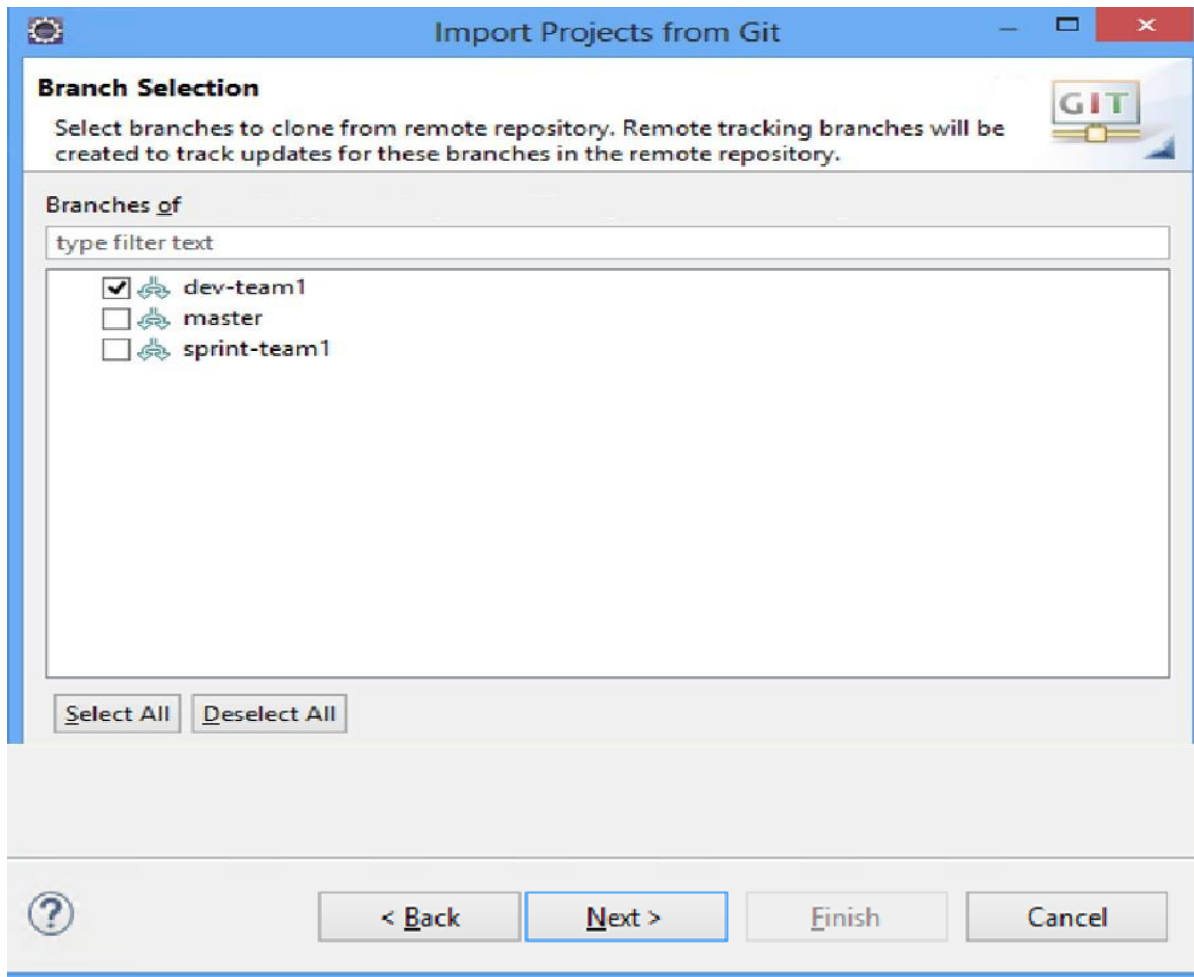
< Back

Next >

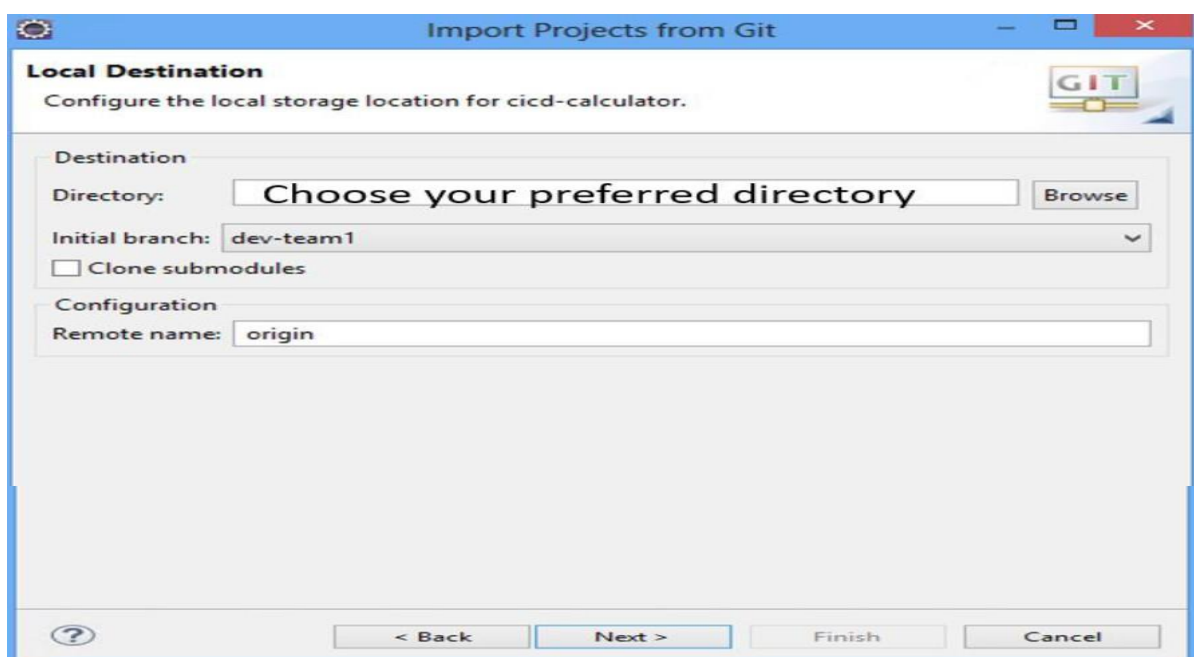
Finish

Cancel

Click Next -> and choose branch (sample shown below)

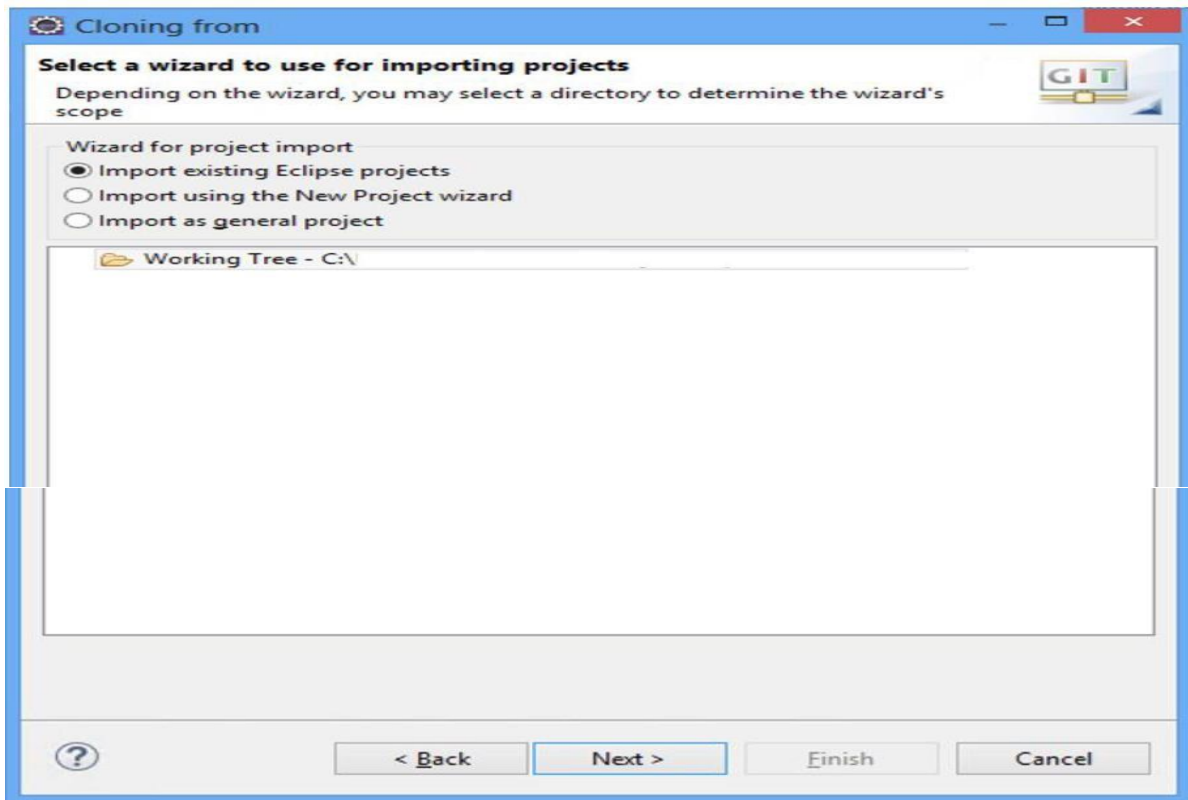


->Click next->browse directory to keep local repository as shown below->

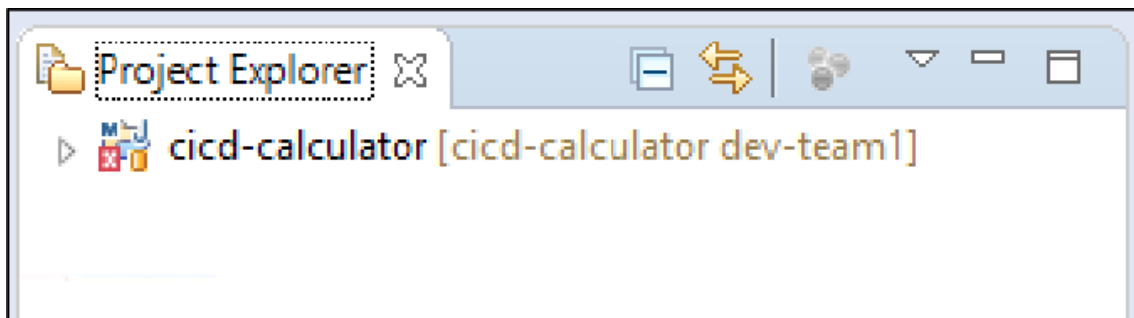


->Next ->Choose first option as shown below->click Next->



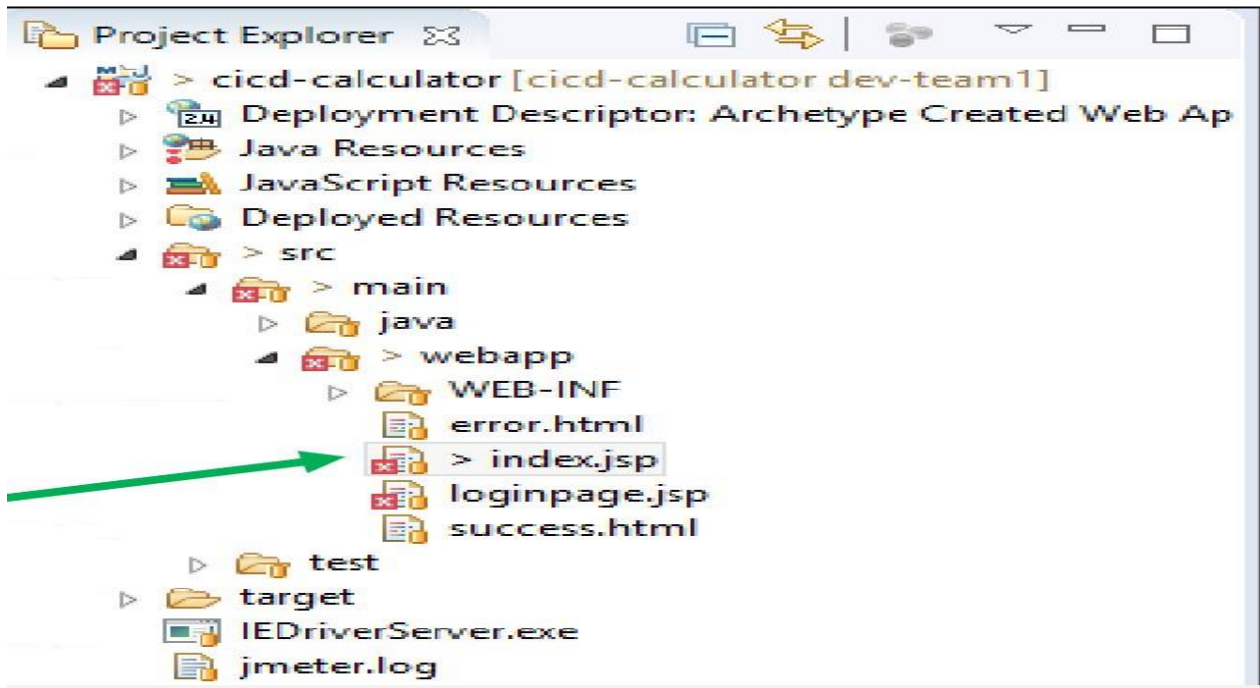


->Click Next -> click Finish->you can observe project explorer view with project cloned as shown below->. The name of the project is JNTU\_Calc\_Application in yourcase.

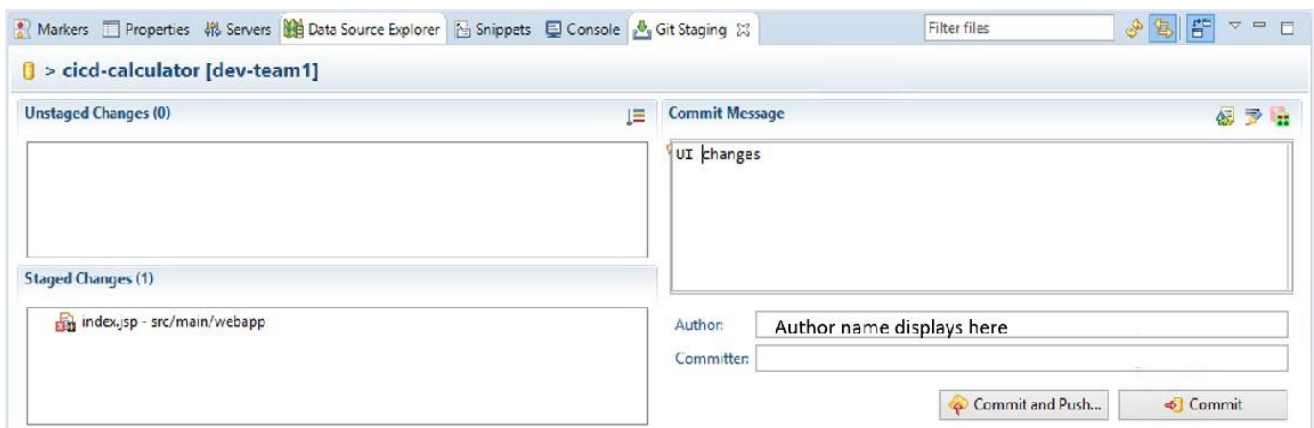


Note:Commit Changes from Eclipse to Local Git Repository

Step 1: Expand the project cicd-calculator-> go to src\main\webapp\index.jsp ->make some changes to the code and save the changes->you can observe “>”symbol on project and edited source file as shown below.



**Step 2:** Right click on project->team->commit->drag the files from Unstaged Changes to Staged Changes ->Enter appropriate comments for the commit as shown below->

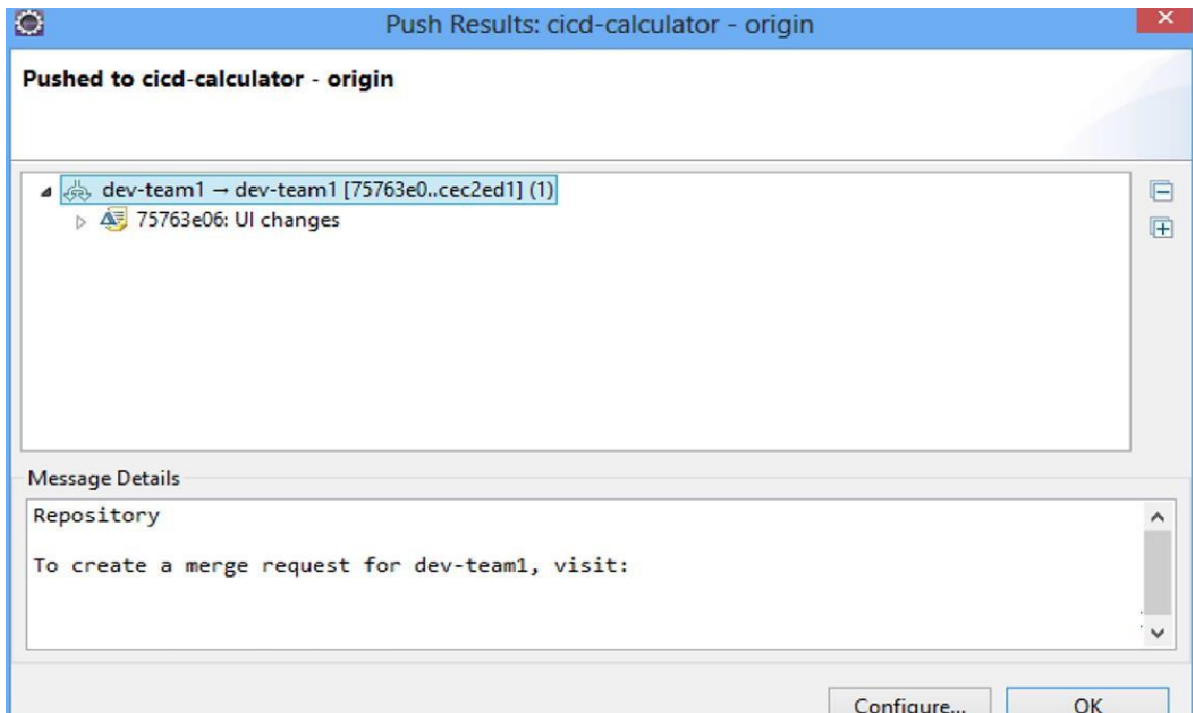


->click on Commit->you can observe the “>” symbol disappearing.  
Note: similar way, we can commit multiple individual changes to local repository using commit option.

## **Push Changes from Local Git Repository (associated with Eclipse) to Remote Git Repository.**

**Step 1:** Right click on project->team-> click on Push to upstream -> you can see changes successfully pushed from local repository to central repository as shown below.

Note: Your repository name and number may be different, this is an illustration



**Note:** if push results to “non fast forward” rejection, perform below mentioned operations in sequence. (This is due to the fact that there has been more changes made possibly by other developers to the central Git repository and hence those changes need to be merged before committing the changes)

Fetch from upstream->merge with local branch ->resolve if there are any merge conflicts->commit-> push.

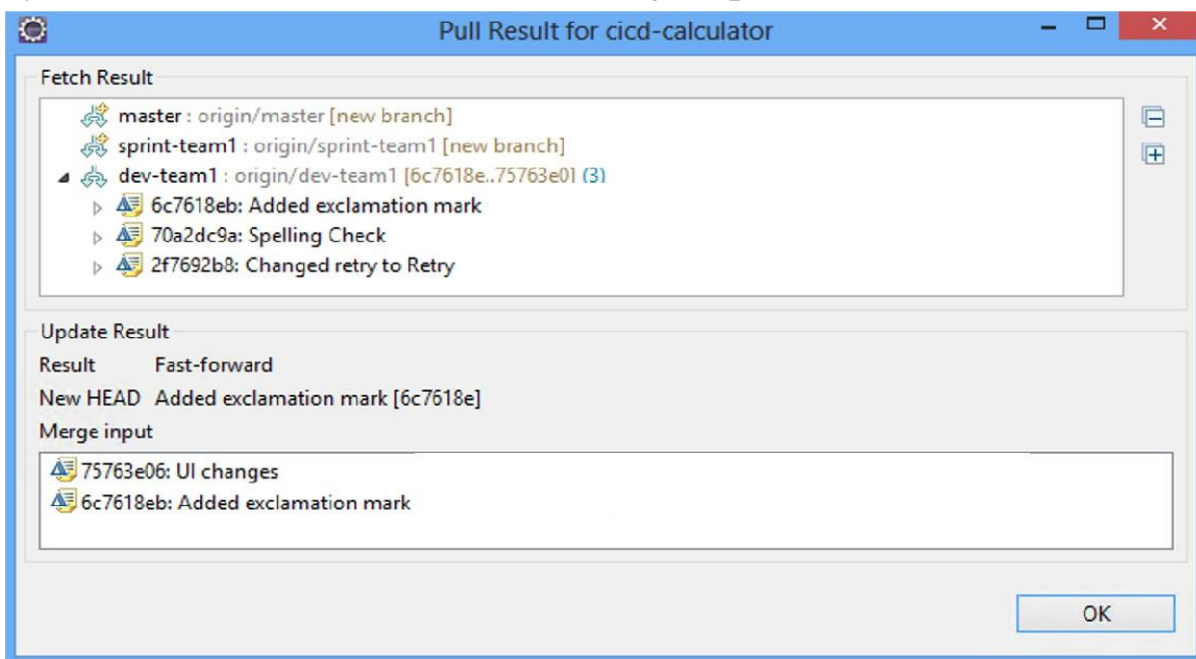
You can observe the revision history using project->team->”Show In History “as shown below.



## **Pull Changes from Remote Git Repository to Local Git Repository (associated with Eclipse).**

Assumption: A team member has committed three changes to remote repository.

**Step 1:** Right click on project->team->Pull->enter credentials if required->you can observe fetch result and merge input see as shown below.



Note: Pull perform two actions in sequence, Fetch and Merge. So if there are any merge conflicts after pull operation, you have to resolve and commit again.

You can observe the updated revision history using project->team->”Show in History”. Pull the project from remote Git repository to local Eclipse

work space using EGit. Practice the following commands-

- a. Commit and push operations
- b. Fetch operation
- c. Merge operation with and without conflicts. When there is a conflict, resolve the conflict and push the code back to the branch provided.

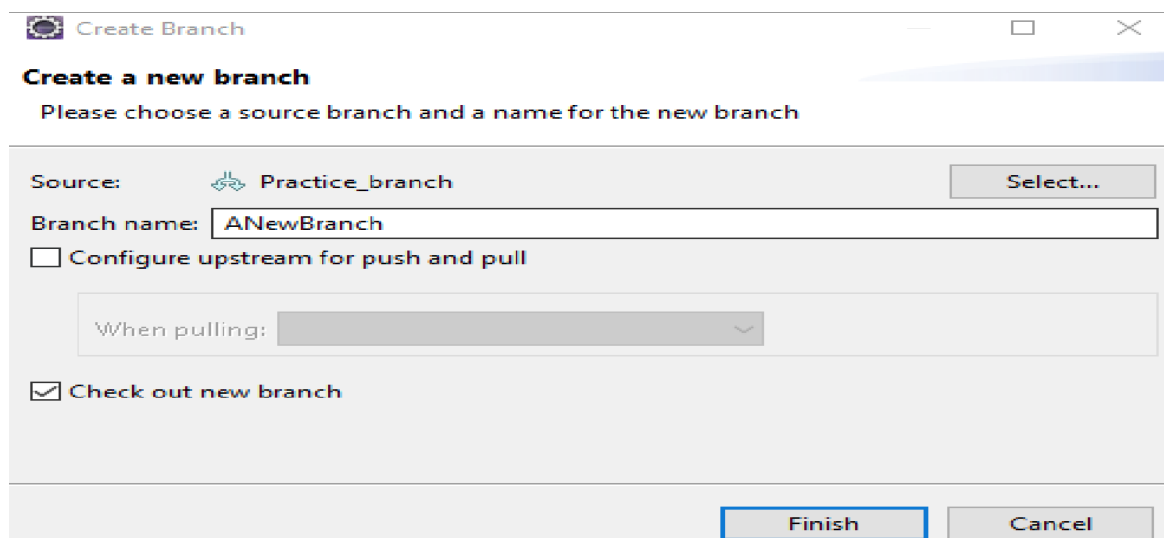
Note: Perform commit, and push operations on remote Git repository by making some changes to source code. You can use the repository provided in the earlier demo for completing this exercise.

### **Creating branches on local repository from eclipse via e-git plugin:**

**Step 1:** Right click on the Project in the Project Explorer and go to Teams -> Switch to -> New Branch...

**Step 2:** The parent branch will by default be that branch that is currently open in Eclipse, you can also change it by clicking on the Select option.

**Step 3:** Name the New Branch and check on the Check out as new branch if you would like to switch to the new branch as well. Click on Finish.



**Summary:** You have learnt to do basic operations with Git related to version control in this exercise.

### **Exercise 3: Creating a project in SonarQube**

**Objective:** Understand creation of project in SonarQube.

**Step 1:** Go to SonarQube URL and login with credentials: admin: password.

**Step 2:** Click on the manually option on the SonarQube dashboard as shown in the screenshot given below.

Are you just testing or have an advanced use-case? Create a project manually.



Manually

**Step 3:** Enter the project display name and project key as calculator as shown in the screenshot given below and click Set Up.

## Create a project

All fields marked with \* are required

**Project display name \***



Up to 255 characters. Some scanners might override the value you provide.

**Project key \***

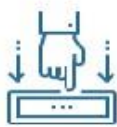


The project key is a unique identifier for your project. It may contain up to 400 characters. Allowed characters are alphanumeric, '-' (dash), '\_' (underscore), '.' (period) and ':' (colon), with at least one non-digit.

Set Up

**Step 4:** Click on Locally option as shown in the screenshot given below.

Are you just testing or have an advanced use-case? Analyze your project locally.



Locally

**Step 5:** Click on the Generate option as shown in the screenshot given below.

Analyze your project

We initialized your project on SonarQube, now it's up to you to launch analyses!

- 1 Provide a token**

Generate a project token

**Token name** ⓘ

Analyze "calculator" Generate

The token is used to identify you when an analysis is performed. If it has been compromised, you can revoke it at any point in time in your user account.
- 2 Run analysis on your project**

**Step 6:** You can see the token generated as shown in the screenshot given below. Click on continue.

**Note:** Copy the token and save it somewhere on your system. It cannot be retrieved later.

Analyze your project

We initialized your project on SonarQube, now it's up to you to launch analyses!

- 1 Provide a token**

Analyze "calculator": Generated token displays here ⓘ

The token is used to identify you when an analysis is performed. If it has been compromised, you can revoke it at any point in time in your user account.

Continue
- 2 Run analysis on your project**

**Step 7:** Select Maven under the run analysis on your project as shown in the screenshot given below.

- 2 Run analysis on your project**

**What option best describes your build?**

Maven Gradle .NET Other (for JS, TS, Go, Python, PHP, ...)

**Step 8:** Paste the copied token in the pom.xml within the <sonar.login> tag as shown in the screenshot given below.

```
<sonar.login>Place the token here</sonar.login>
```



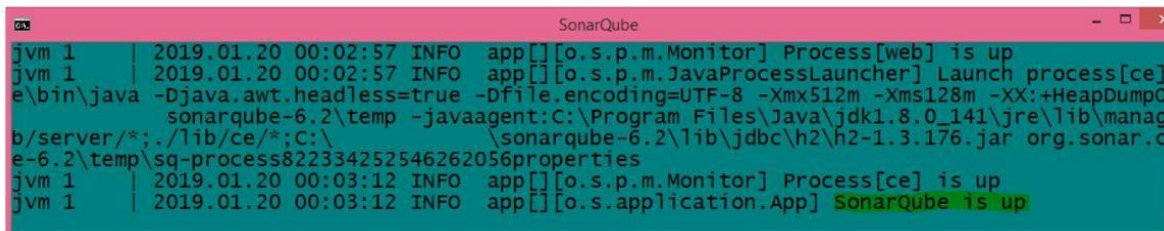
**Summary:** You have learnt to create a project in sonarqube in this exercise.

## **Exercise 4: Using Sonarqube with Sonar-Runner.**

**Objective:** Understand running of Sonarqube using Sonar-Runner command-line tool.

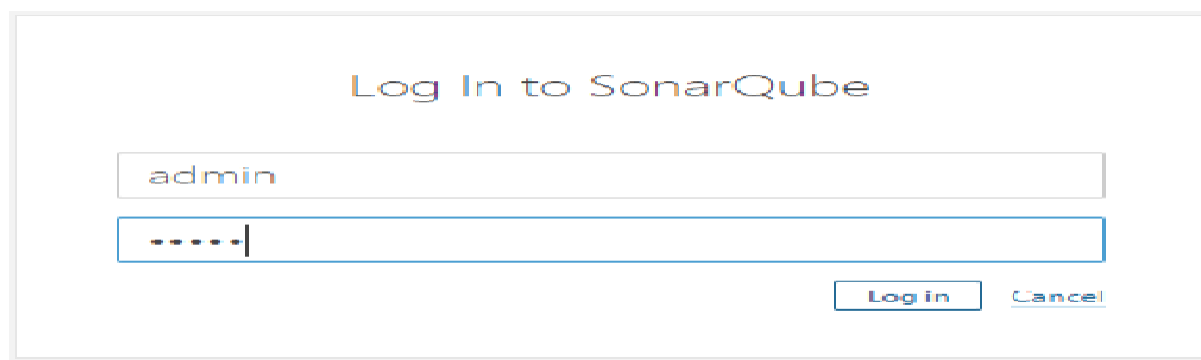
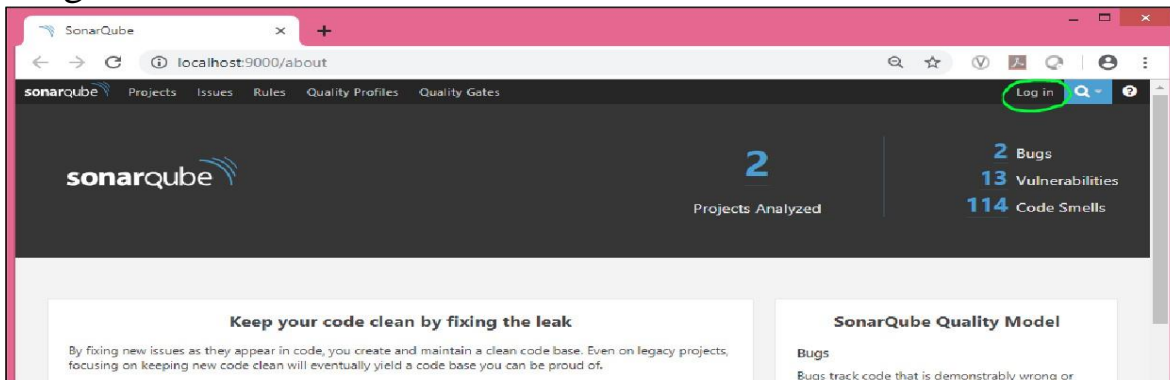
**Step 1:** Start Sonar server by using the appropriate bat file.

Once started, you should see success message in console window:



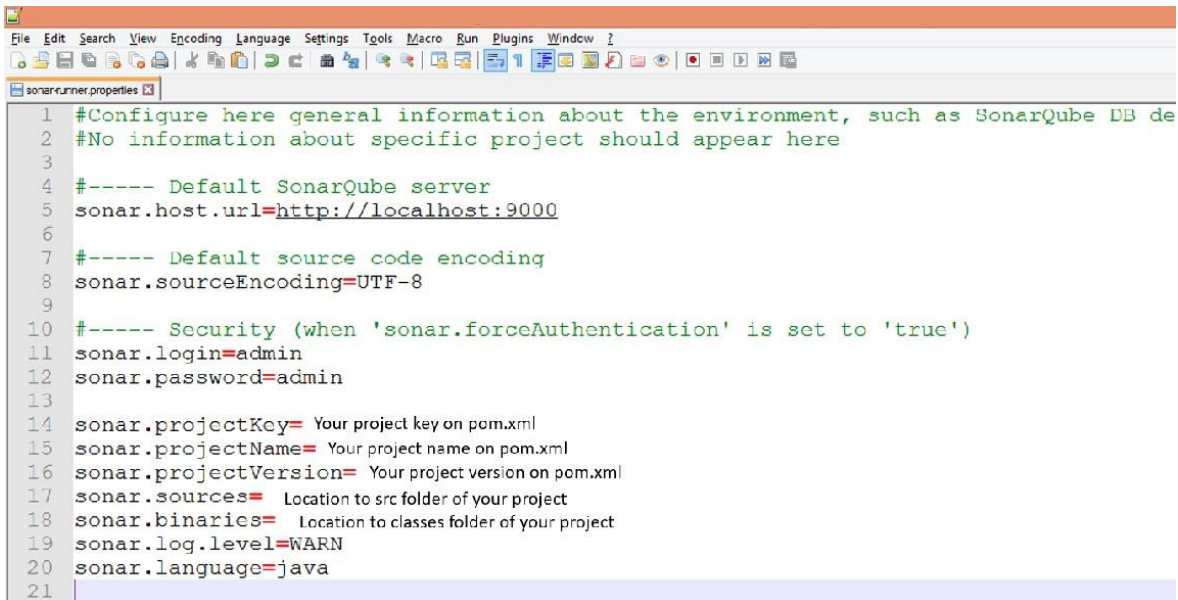
```
jvm 1 | 2019.01.20 00:02:57 INFO app[] [o.s.p.m.Monitor] Process[web] is up
jvm 1 | 2019.01.20 00:02:57 INFO app[] [o.s.p.m.JavaProcessLauncher] Launch process[ce]
e\bin\java -Djava.awt.headless=true -Dfile.encoding=UTF-8 -Xmx512m -Xms128m -XX:+HeapDumpOn
sonarqube-6.2\temp -javaagent:C:\Program Files\Java\jdk1.8.0_141\jre\lib\manag
e-6.2\temp\sq-process822334252546262056properties
jvm 1 | 2019.01.20 00:03:12 INFO app[] [o.s.p.m.Monitor] Process[ce] is up
jvm 1 | 2019.01.20 00:03:12 INFO app[] [o.s.application.App] SonarQube is up
```

Go to SonarQube server dashboard at <http://localhost:9000> and login using admin/admin



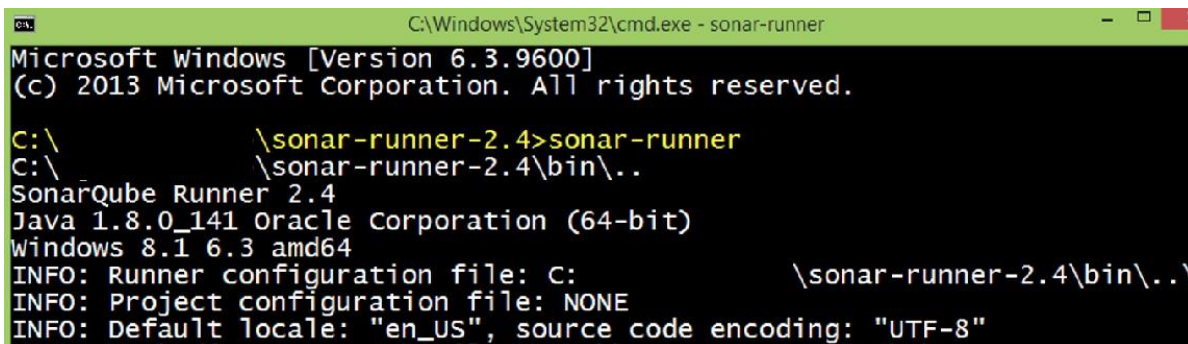
**Step 2:** Go to conf folder of Sonar Runner. Make the following changes based on the name of the Project provided to you and the path where it exists in your system:



A screenshot of an IDE window titled 'sonar-runner.properties'. The file contains configuration for SonarQube. The content is as follows:

```
1 #Configure here general information about the environment, such as SonarQube DB de
2 #No information about specific project should appear here
3
4 #----- Default SonarQube server
5 sonar.host.url=http://localhost:9000
6
7 #----- Default source code encoding
8 sonar.sourceEncoding=UTF-8
9
10 #----- Security (when 'sonar.forceAuthentication' is set to 'true')
11 sonar.login=admin
12 sonar.password=admin
13
14 sonar.projectKey= Your project key on pom.xml
15 sonar.projectName= Your project name on pom.xml
16 sonar.projectVersion= Your project version on pom.xml
17 sonar.sources= Location to src folder of your project
18 sonar.binaries= Location to classes folder of your project
19 sonar.log.level=WARN
20 sonar.language=java
21
```

Open command prompt inside the project src folder in File Explorer and run the following command.

A screenshot of a Windows command prompt window titled 'C:\Windows\System32\cmd.exe - sonar-runner'. The output shows the execution of the sonar-runner command. The content is as follows:

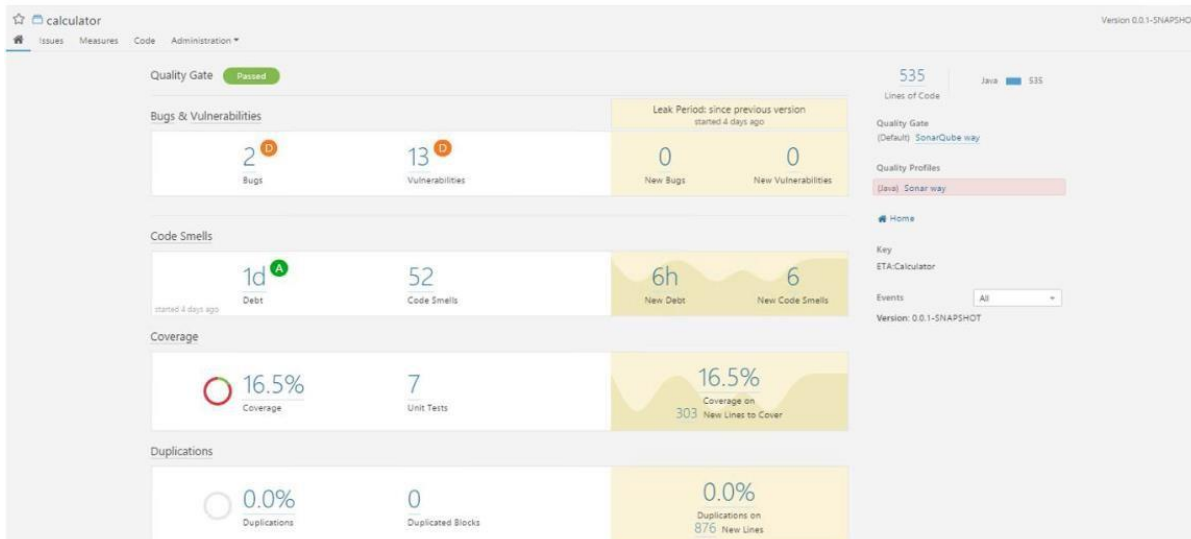
```
C:\Windows\System32\cmd.exe - sonar-runner
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\> \sonar-runner-2.4>sonar-runner
C:\> \sonar-runner-2.4\bin\..
SonarQube Runner 2.4
Java 1.8.0_141 Oracle Corporation (64-bit)
Windows 8.1 6.3 amd64
INFO: Runner configuration file: C:
INFO: Project configuration file: NONE
INFO: Default locale: "en_US", source code encoding: "UTF-8"
```

Once execution is successful:

- a) Observe the static code analysis report and critical issues on SonarQube dashboard. Select your project name from list of projects to view the latest report.
- b) Resolve the technical issues in code and rerun the Maven build. Notice the changes to the technical debt value.

Run the build file and observe the results.



Summary of this Exercise:

You have learnt to observe the results of static code analysis using Sonarqube.

## **Exercise 5: Creating a local repository in Artifactory**

**Objective:** Understand creation of local repository in Artifactory

**Step 1:** Go to Artifactory URL and login with credentials: admin: Password1!

**Step 2:** Go to Administration -> Repositories -> Repositories -> Add repositories -> Local Repository.

**Step 3:** Select the package type as Maven

**Step 4:** To add the repository key, go to pom.xml and copy the <name> tag value(Calc\_Dev\_Snapshot) as shown in the screenshot given below.

```
<distributionManagement>
  <repository>
    <id>artifactory</id>
    <name>Calc_Dev_Snapshot</name>
    <url>http://localhost:8081/artifactory/Calc_Dev_Snapshot</url>
  </repository>
</distributionManagement>
```

**Step 5:** Click on Create Local Repository.

**Step 6:** You can view the binaries stored in the Artifactory under Application -> Artifactory -> Packages.

**Summary:** You have learnt to create a local repository in Artifactory in this exercise.

# Build Automation: Maven

## **Exercise 6: Build automation using Maven**

Objective: Understand build automation by writing a script in Maven with goals to invoke activities in a CI pipeline.

```
<?xml version="1.0" encoding="UTF-8"?>
<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>ETA</groupId>
  <artifactId>Calculator</artifactId>
  <packaging>war</packaging>
  <version>0.0.1-SNAPSHOT</version>
  <name>calculator</name>
  <url>http://calculator</url>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.11</version>
    </dependency>
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>javax.servlet-api</artifactId>
      <version>3.1.0</version>
    </dependency>
    <dependency>
      <groupId>org.seleniumhq.selenium</groupId>
      <artifactId>selenium-java</artifactId>
      <version>3.6.0</version>
      <scope>provided</scope>
    </dependency>
  </dependencies>

  <build>
    <finalName>calculator</finalName>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-war-plugin</artifactId>
        <version>2.1.1</version>
        <configuration>
          <archive>
            <manifestEntries>

<version>${project.version}</version>
```

```

        </manifestEntries>
    </archive>
</configuration>
</plugin>
<plugin>
    <groupId>org.sonarsource.scanner.maven</groupId>
    <artifactId>sonar-maven-plugin</artifactId>
    <version>3.2</version>
</plugin>
<plugin>
    <groupId>org.jacoco</groupId>
    <artifactId>jacoco-maven-plugin</artifactId>
    <version>0.7.9</version>
    <executions>
        <execution>
            <id>default-prepare-agent</id>
            <goals>
                <goal>prepare-agent</goal>
            </goals>
        </execution>
        <execution>
            <id>default-report</id>
            <phase>prepare-package</phase>
            <goals>
                <goal>report</goal>
            </goals>
        </execution>
        <execution>
            <id>default-check</id>
            <goals>
                <goal>check</goal>
            </goals>
            <configuration>
                <rules>
                    <!-- implementation is
needed only for Maven 2 -->
                    <rule
implementation="org.jacoco.maven.RuleConfiguration">
                        <element>BUNDLE</element>
                        <limits>
                            <!--
implementation is needed only for Maven 2 -->
                            <limit
implementation="org.jacoco.report.check.Limit">
                                <counter>COMPLEXITY</counter>
                                <value>COVEREDRATIO</value>
                                <minimum>0.10</minimum>
                            </limit>
                        </limits>
                    </rule>
                </rules>

```

```

        </configuration>
      </execution>
    </executions>
  </plugin>
</plugins>
</build>

<profiles>
  <profile>
    <id>ut</id>
    <build>
      <plugins>
        <plugin>

          <groupId>org.apache.maven.plugins</groupId>
          <artifactId>maven-surefire-
plugin</artifactId>

          <configuration>
            <includes>

<include>**/Calclaterut.java</include>

          </includes>
        </configuration>
      </plugin>
    </plugins>
  </build>
</profile>
<profile>
  <id>it</id>
  <build>
    <plugins>
      <plugin>

        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-surefire-
plugin</artifactId>

        <configuration>
          <includes>

<include>**/CalculatorIT.java</include>

        </includes>
      </configuration>
    </plugin>
  </plugins>
</build>
</profile>
<profile>
  <id>pt</id>
  <build>
    <plugins>
      <plugin>

```

```

        <groupId>com.lazerycode.jmeter</groupId>
        <artifactId>jmeter-maven-
plugin</artifactId>
        <version>2.4.0</version>
        <executions>
            <execution>
                <id>jmeter-tests</id>
                <phase>test</phase>
                <goals>
                    <goal>jmeter</goal>
                </goals>
            </execution>
        </executions>
    </plugin>
</plugins>
</build>
</profile>
</profiles>

</project>

<!-- -->

```

Once you run the pom.xml in the order “clean compile test jacoco:report sonar:sonar war:war”, you can see the output on console as shown below.

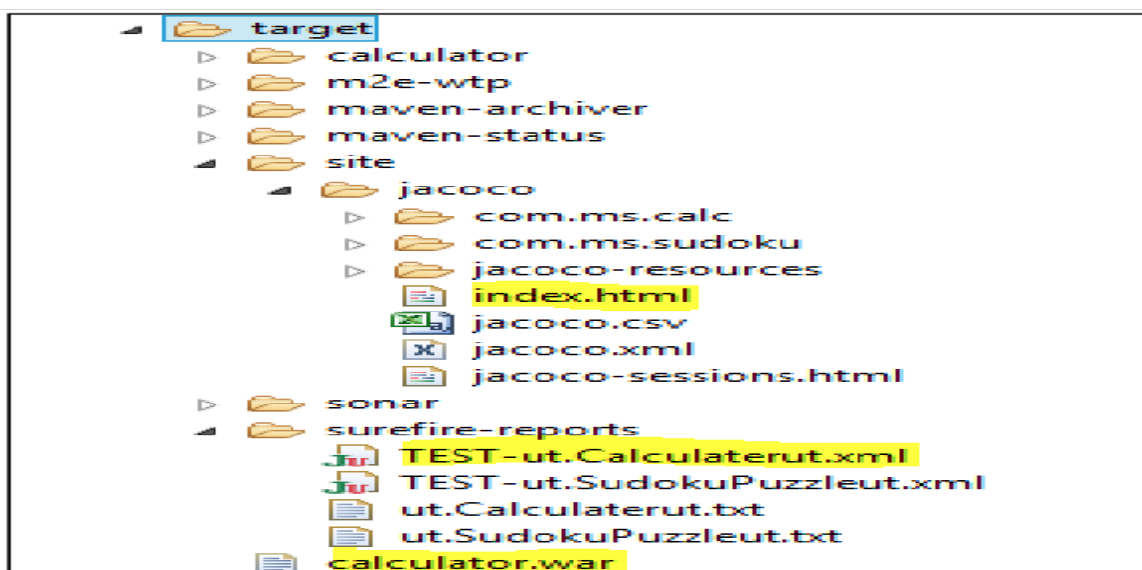
```

[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 48.720 s
[INFO] Finished at:
[INFO] Final Memory: 40M/351M
[INFO] -----

```

You can also find the results of the maven goals executed in the target folder of your project:

1. JUnit test reports in xml and txt: sure fire-reports folder
2. Jacoco code coverage reports in html and xml: Under site/Jacoco/
3. War file.



Summary of this Exercise: You have learnt to use Maven tool for build automation.

# Continuous Integration: Jenkins

## Exercise 7: Jenkins Installation & System Configuration.

**Objective:** Configure Jenkins for CI

Note:

You can install Jenkins in two ways on Windows –

- a. Install Jenkins as a Windows service
- b. Use webserver with a servlet container like Glass Fish or Tomcat, and then deploy Jenkins. War to it.

We have used the first approach in this exercise.

### Configuring Jenkins

**Step 1:** Start the Jenkins server and enter URL `http://localhost:8064/jenkins` in browser.

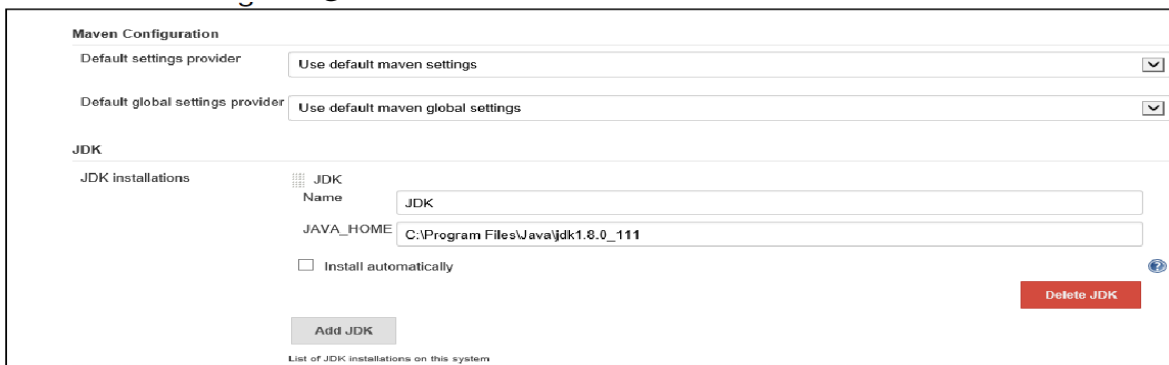
**Step 2:** Go to  [Manage Jenkins](#) ->  [Configure System](#)  
Configure global settings and paths.

**Step 3:** You can observe Jenkins Maven integration as shown below:



### Additional configurations

1. Provide the tool configuration (JDK, Maven) in the Manage Jenkins -> Global Tool configuration tab.



List of JDK installations on this system

### Git

Git installations

Git
<div> <div>Name</div> <div>Default</div> </div> <div> <div>Path to Git executable</div> <div>git.exe</div> </div> <div> <input type="checkbox"/> Install automatically         </div>

[Delete Git](#)

[Add Git](#)

## 2. Go to global properties

(Manage Jenkins->Configure system->Global properties->environment variables) and set JAVA\_HOME and M2\_HOME to the respective machine path.

Summary of this Exercise:

You have learnt to configure Jenkins.

## **Exercise 8: Download the plugins in Jenkins**

**Objective:** Download the plugins in Jenkins

**Step 1:** Go to Jenkins URL

**Step 2:** Go to Manage Jenkins -> Manage Plugins from the Jenkins dashboard

Step 3: Under the available tab of plugins manager search for the copy artefact plugin and check the check box as shown in the screenshot given below.

**Plugin Manager**

Q. copy artifact

Updates Available Installed Advanced

Install	Name	Released
<input checked="" type="checkbox"/>	<b>Copy Artifact 1.46.4</b> Build Parameters Build Tools Adds a build step to copy artifacts from another project.	2 mo 2 days ago

[Install without restart](#)
[Download now and install after restart](#)
 Update information obtained: 2 days 4 hr ago
 [Check now](#)



Step 4: Repeat the step – 3 and select the below mentioned plugins and click on install without restart.

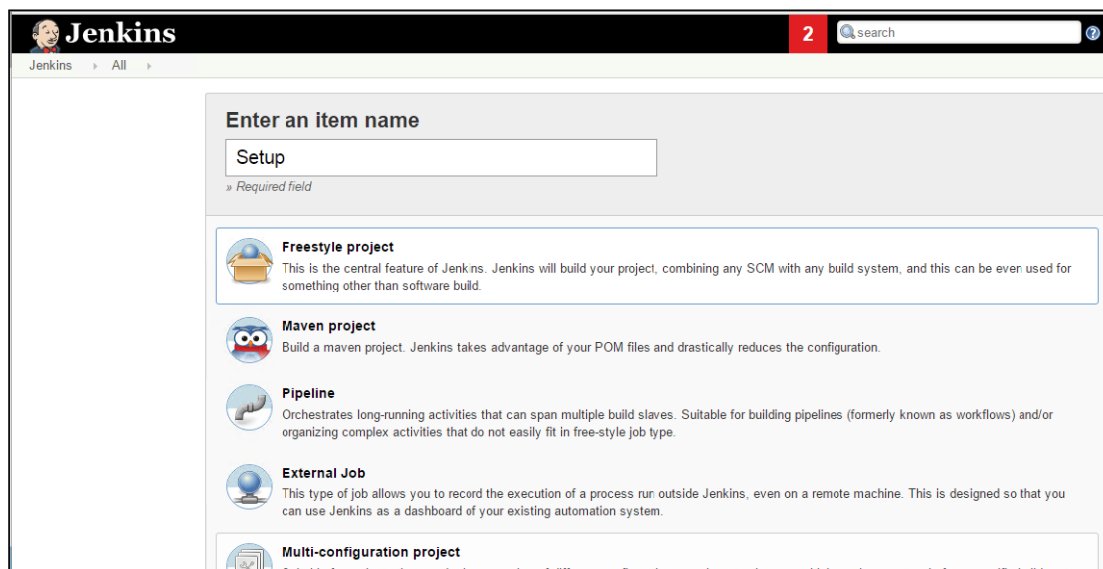
- a. Jacoco
- b. Artifactory
- c. Build pipeline
- d. Deploy to container

Summary: You have learnt to download plugins in Jenkins in this exercise.

## **Exercise 9: Creating Central CI pipeline**

**Objective:** Creating main line CI pipeline

Create a new Folder item with name “Central\_CI” using “New Item” option as shown below. Add jobs of type Freestyle Project for each of tasks needed in the the continuous integration pipeline.



## **Step 1:**Create below mentioned job – Setup

The screenshot displays the Jenkins job configuration interface. The top navigation bar includes tabs for General, Source Code Management, Build Triggers, Build Environment, Build, and Post-build Actions. The General tab is active, showing a description field with the text "code check-out". Below this, there is a checkbox for "Discard old builds" which is checked. The "Strategy" dropdown is set to "Log Rotation". Under "Log Rotation", there are two input fields: "Days to keep builds" set to 3 and "Max # of builds to keep" set to 3. A "Save" button is visible. Below the "Discard old builds" section, there are several checkboxes: "GitHub project", "This project is parameterized", "Throttle builds", "Disable this project", "Execute concurrent builds if necessary", and "Restrict where this project can be run". A "Save" button is also present. The "Source Code Management" tab is partially visible, showing options for "None" and "Git". Under "Git", there are fields for "Repository URL" and "Credentials", both with "Enter your repo URL" and "Enter Credentials" respectively. There is also a "Branches to build" section with a "Branch Specifier (blank for 'any')" field set to "Enter your working branch". A "Repository browser" dropdown is set to "(Auto)". An "Additional Behaviours" section has an "Add" button. A "Subversion" option is also visible at the bottom.

**General** Source Code Management Build Triggers Build Environment Build Post-build Actions

Description code check-out

[Plain text] Preview

☒ Discard old builds

Strategy Log Rotation

Days to keep builds 3

If not empty, build records are only kept up to this number of days

Max # of builds to keep 3

If not empty, only up to this number of build records are kept

Advanced...

☐ GitHub project

☒ Permission to Copy Artifact

Save Apply

☐ This project is parameterized

☐ Throttle builds

☐ Disable this project

☐ Execute concurrent builds if necessary

☐ Restrict where this project can be run

Advanced...

**Source Code Management**

☐ None

☒ Git

Repositories

Repository URL Enter your repo URL

Credentials Enter Credentials Add

Advanced...

Add Repository

Branches to build

Branch Specifier (blank for 'any') Enter your working branch

Add Branch

Repository browser (Auto)

Additional Behaviours Add

☐ Subversion

## Build Triggers

- ☐ Trigger builds remotely (e.g., from scripts)
- ☐ Build after other projects are built
- ☐ Build periodically
- ☐ Build when another project is promoted
- ☐ GitHub hook trigger for GITScm polling
- ☒ Poll SCM

Schedule

H/2 \* \* \* \*

Ignore post-commit hooks ☐

## Build Environment

- ☒ Delete workspace before build starts

Advanced...

Abort the build if it's stuck

Save

Apply

Use secret text(s) or file(s)

Abort the build if it's stuck

Save

Apply

Use secret text(s) or file(s)

With Ant

## Build

Invoke top-level Maven targets

Maven Version

new

Goals

clean

Advanced...

Add build step

## Post-build Actions

Save

Apply

Files to archive

\*\*\*

Advanced...

Build other projects

Projects to build

Compile

☒ Trigger only if build is stable

☐ Trigger even if the build is unstable

☐ Trigger even if the build fails

Add post-build action

Save

Apply

## Step 2: Create below mentioned job – Compile

The screenshot displays the Jenkins job configuration interface for a job named 'Compile'. The interface is divided into several tabs: General, Source Code Management, Build Triggers, Build Environment, Build, and Post-build Actions. The 'General' tab is currently selected.

**General Tab:**

- Description:** A text area containing the word 'compiling'.
- Discard old builds:** A checkbox that is checked. Below it, the 'Strategy' is set to 'Log Rotation'. Two input fields are present: 'Days to keep builds' (set to 3) and 'Max # of builds to keep' (set to 3). Both fields have a note: 'if not empty, build records are only kept up to this number of days'.
- Advanced...** button.
- GitHub project:** A checkbox that is unchecked.
- Permission to Copy Artifact:** A checkbox that is unchecked. Below it are 'Save' and 'Apply' buttons.
- Advanced...** button.
- Other options:** A list of checkboxes: 'This project is parameterized', 'Throttle builds', 'Disable this project', 'Execute concurrent builds if necessary', and 'Restrict where this project can be run'. All are unchecked.
- Advanced...** button.

**Source Code Management Tab:**

- SCM:** A dropdown menu with 'None' selected. Other options are 'Git' and 'Subversion'.
- Advanced...** button.

**Build Triggers Tab:**

- Build Triggers:** A section with 'Save' and 'Apply' buttons. Below them are several checkboxes: 'Build after other projects are built', 'Build periodically', 'Build when another project is promoted', 'GitHub hook trigger for GITScm polling', and 'Poll SCM'. All are unchecked.
- Advanced...** button.

**Build Environment Tab:**

- Delete workspace before build starts:** A checkbox that is checked.
- Advanced...** button.
- Other options:** A list of checkboxes: 'Abort the build if it's stuck', 'Add timestamps to the Console Output', and 'Use secret text(s) or file(s)'. All are unchecked.
- Advanced...** button.

Copy artifacts from another project

Project name

Setup

Which build

Latest successful build

☐ Stable build only

Artifacts to copy

\*\*/\*

Artifacts not to copy

Target directory

Parameter filters

☐ Flatten directories

☐ Optional

☒ Fingerprint Artifacts

Advanced...

Save

Apply

Invoke top-level Maven targets

Maven Version

new

Goals

compile

Advanced...

Add build step

Post-build Actions

Archive the artifacts

Files to archive

\*\*/\*

Advanced...

Build other projects

Projects to build

StaticAnalysis

☒ Trigger only if build is stable

☐ Trigger even if the build is unstable

☐ Trigger even if the build fails

Add post-build action

Save

Apply

### **Step 3:**Create below mentioned job – Unit test

GeneralSource Code ManagementBuild TriggersBuild EnvironmentBuildPost-build Actions

DescriptionunitTest

[Plain text] Preview

☒ Discard old builds

StrategyLog Retention

Days to keep builds3

if not empty, build records are only kept up to this number of days

Max # of builds to keep3

if not empty, only up to this number of build records are kept

Advanced...

☐ GitHub project

☐ Permission to Copy Artifact

☐ Promote builds when...

☐ This project is parameterized

☐ Throttle builds

☐ Disable this project

☐ Execute concurrent builds if necessary

☐ Restrict where this project can be run

Advanced...

Source Code Management

☒ None

☐ Git

☐ Subversion

Build Triggers

☐ Trigger builds remotely (e.g., from scripts)

☐ Build after other projects are built

☐ Build periodically

☐ Build when another project is promoted

☐ GitHub hook trigger for GITScm polling

☐ Poll SCM

Build Environment

☒ Delete workspace before build starts

Advanced...

☐ Abort the build if it's stuck

☐ Add timestamps to the Console Output

☐ Use secret text(s) or file(s)

☐ With Ant

## Build

### Copy artifacts from another project

Project name

Which build

☐ Stable build only

Artifacts to copy

Artifacts not to copy

Target directory

Parameter filters

☐ Flatten directories ☐ Optional ☒ Fingerprint Artifacts

Advanced...

### Invoke top-level Maven targets

Maven Version

Goals

Advanced...

Add build step

## Post-build Actions

### Archive the artifacts

Files to archive

Advanced...

### Build other projects

Projects to build

☒ Trigger only if build is stable

☐ Trigger even if the build is unstable

☐ Trigger even if the build fails

Add post-build action

Save

Apply

## Step 4: Create below mentioned job – Code coverage

The screenshot shows the Jenkins job configuration for a job named 'codeCoverage'. The 'General' tab is selected, showing options for 'Discard old builds' (checked) with a 'Log Rotation' strategy, 'Days to keep builds' set to 3, and 'Max # of builds to keep' set to 3. Below these are several checkboxes for project settings like 'GitHub project', 'Permission to Copy Artifact', and 'Throttle builds'. The 'Source Code Management' section is partially visible at the bottom.

**General** | Source Code Management | Build Triggers | Build Environment | Build | Post-build Actions

Description: codeCoverage

[Plain text] [Preview](#)

☒ Discard old builds

Strategy: Log Rotation

Days to keep builds: 3  
if not empty, build records are only kept up to this number of days

Max # of builds to keep: 3  
if not empty, only up to this number of build records are kept

[Advanced...](#)

☐ GitHub project  
☐ Permission to Copy Artifact  
☐ Promote builds when...  
☐ This project is parameterized  
☐ Throttle builds  
☐ Disable this project  
☐ Execute concurrent builds if necessary  
☐ Restrict where this project can be run

[Advanced...](#)

**Source Code Management**

☒ None

The screenshot shows the 'Build Triggers' and 'Build Environment' sections of the Jenkins job configuration. The 'Build Triggers' section includes checkboxes for 'Trigger builds remotely', 'Build after other projects are built', 'Build periodically', 'Build when another project is promoted', 'GitHub hook trigger for GITScm polling', and 'Poll SCM'. The 'Build Environment' section includes a checked checkbox for 'Delete workspace before build starts' and an 'Advanced...' button. Below this are checkboxes for 'Abort the build if it's stuck', 'Add timestamps to the Console Output', 'Use secret text(s) or file(s)', and 'With Ant'.

☐ Git  
☐ Subversion

**Build Triggers**

☐ Trigger builds remotely (e.g., from scripts)  
☐ Build after other projects are built  
☐ Build periodically  
☐ Build when another project is promoted  
☐ GitHub hook trigger for GITScm polling  
☐ Poll SCM

**Build Environment**

☒ Delete workspace before build starts

[Advanced...](#)

☐ Abort the build if it's stuck  
☐ Add timestamps to the Console Output  
☐ Use secret text(s) or file(s)  
☐ With Ant



### Build

Copy artifacts from another project

Project name

UnitTest

Which build

Latest successful build

Stable build only

Artifacts to copy

\*\*/\*

Artifacts not to copy

Target directory

Parameter filters

Flatten directories

Optional

Fingerprint Artifacts

Advanced...

Invoke top-level Maven targets

Maven Version

new

Goals

jacoco:report

Advanced...

Add build step

### Post-build Actions

Archive the artifacts

Files to archive

\*\*/\*

Advanced...

Build other projects

Projects to build

War

Trigger only if build is stable

Trigger even if the build is unstable

Trigger even if the build fails

Add post-build action

Save

Apply

DevOps-Continuous Integration and Continuous Delivery

## Step 5: Create below mentioned job – Static analysis

General

Source Code Management

Build Triggers

Build Environment

Build

Post-build Actions

Description

static analysis sonar

[Plain text] [Preview](#)

☒ Discard old builds

Strategy

Log Rotation

Days to keep builds

3

if not empty, build records are only kept up to this number of days

Max # of builds to keep

3

if not empty, only up to this number of build records are kept

Advanced...

☐ GitHub project

☐ Permission to Copy Artifact

☐ Promote builds when...

☐ This project is parameterized

☐ Throttle builds

☐ Disable this project

☐ Execute concurrent builds if necessary

☐ Restrict where this project can be run

Advanced...

Source Code Management

☒ None

☐ Git

☐ Subversion

Build Triggers

☐ Trigger builds remotely (e.g., from scripts)

☐ Build after other projects are built

☐ Build periodically

☐ Build when another project is promoted

☐ GitHub hook trigger for GITScm polling

☐ Poll SCM

Build Environment

☒ Delete workspace before build starts

Advanced...

☐ Abort the build if it's stuck

☐ Add timestamps to the Console Output

☐ Use secret text(s) or file(s)

☐ With Ant

## Build

### Copy artifacts from another project

Project name  ⓘ

Which build  ⓘ

☐ Stable build only

Artifacts to copy  ⓘ

Artifacts not to copy  ⓘ

Target directory  ⓘ

Parameter filters  ⓘ

☐ Flatten directories ☐ Optional ☒ Fingerprint Artifacts ⓘ

Advanced...

### Invoke top-level Maven targets

Maven Version  ⓘ

Goals  ⓘ

Advanced...

Add build step ▾

## Post-build Actions

### Archive the artifacts

Files to archive  ⓘ

Files to archive  ⓘ

Advanced...

### Build other projects

Projects to build  ⓘ

☒ Trigger only if build is stable

☐ Trigger even if the build is unstable

☐ Trigger even if the build fails

Add post-build action ▾

Save

Apply

## Step 6: Create below mentioned job – war

General

Source Code Management

Build Triggers

Build Environment

Build

Post-build Actions

Description

war

[Plain text] Preview

☒ Discard old builds

Strategy

Log Rotation

Days to keep builds

3

if not empty, build records are only kept up to this number of days

Max # of builds to keep

3

if not empty, only up to this number of build records are kept

Advanced...

☐ GitHub project

☐ Permission to Copy Artifact

☐ Promote builds when...

☐ This project is parameterized

☐ Throttle builds

☐ Disable this project

☐ Execute concurrent builds if necessary

☐ Restrict where this project can be run

Advanced...

Source Code Management

None

Git

Subversion

Build Triggers

☐ Trigger builds remotely (e.g., from scripts)

☐ Build after other projects are built

☐ Build periodically

☐ Build when another project is promoted

☐ GitHub hook trigger for GITSCM polling

☐ Poll SCM

Build Environment

☒ Delete workspace before build starts

Advanced...

☐ Abort the build if it's stuck

☐ Add timestamps to the Console Output

☐ Use secret text(s) or file(s)

☐ With Ant

## Build Environment

☒ Delete workspace before build starts

Advanced...

☐ Abort the build if it's stuck

☐ Add timestamps to the Console Output

☐ Use secret text(s) or file(s)

☐ With Ant



## Build

### Copy artifacts from another project

Project name: CodeCoverage

Which build: Latest successful build

☐ Stable build only

Artifacts to copy: \*\*/\*

Artifacts not to copy:

Target directory:

Parameter filters:

☐ Flatten directories

☐ Optional

☒ Fingerprint Artifacts

Advanced...

### Invoke top-level Maven targets

Maven Version: new

Goals: war:war

Advanced...

Add build step

### Build other projects

Projects to build: ToArtifactory

☒ Trigger only if build is stable

☐ Trigger even if the build is unstable

☐ Trigger even if the build fails

Add post-build action

Save

Apply

## Step 7: Create below mentioned job – To Artifactory

General

Source Code Management

Build Triggers

Build Environment

Build

Post-build Actions

Description

war

[Plain text] Preview

☒ Discard old builds

Strategy

Log Rotation

Days to keep builds

3

If not empty, build records are only kept up to this number of days

Max # of builds to keep

3

If not empty, only up to this number of build records are kept

Advanced...

☐ GitHub project

☐ Permission to Copy Artifact

☐ Promote builds when...

☐ This project is parameterized

☐ Throttle builds

☐ Disable this project

☐ Execute concurrent builds if necessary

☐ Restrict where this project can be run

Advanced...

Source Code Management

None

Git

Subversion

Build Triggers

☐ Trigger builds remotely (e.g., from scripts)

☐ Build after other projects are built

☐ Build periodically

☐ Build when another project is promoted

☐ GitHub hook trigger for GITScm polling

☐ Poll SCM

Build Environment

☒ Delete workspace before build starts

Advanced...

☐ Abort the build if it's stuck

☐ Add timestamps to the Console Output

☐ Use secret text(s) or file(s)

☐ With Ant

**Build Environment**

- ☒ Delete workspace before build starts
  - Advanced...
- ☐ Abort the build if it's stuck
- ☐ Add timestamps to the Console Output
- ☐ Use secret key(s) or file(s)
- ☐ With Ant

**Build**

☐ Copy artifacts from another project

Project name:

Which build:

☐ Stable build only

Artifacts to copy:

Artifacts not to copy:

Target directory:

Parameter filters:

☐ Flatten directories ☐ Optional ☒ Fingerprint Artifacts

Advanced...

☐ Invoke top-level Maven targets

Maven Version:

Goals:

Advanced...

Add build step ▾

**Post-build Actions**

☐ Archive the artifacts

Files to archive:

Advanced...

☐ Build other projects

Projects to build:

☒ Trigger only if build is stable

## Step 8: Create below mentioned job – SmokeTest

**General** Source Code Management Build Triggers Build Environment Build Post-build Actions

Description:

(Plain text) [Preview](#)

☒ Discard old builds

Strategy:

Days to keep builds:

if not empty, build records are only kept up to this number of days

Max # of builds to keep:

if not empty, only up to this number of build records are kept

Advanced...

- ☐ Github project
- ☐ Permission to Copy Artifact
- ☐ Promote builds when...
- ☐ This project is parameterized
- ☐ Throttle builds
- ☐ Disable this project
- ☐ Execute concurrent builds if necessary
- ☐ Restrict where this project can be run

Advanced...

**Source Code Management**

\* None

☐ Git

☐ Subversion

**Build Triggers**

- ☐ Trigger builds remotely (e.g., from scripts)
- ☐ Build after other projects are built
- ☐ Build periodically
- ☐ Build when another project is promoted
- ☐ Github hook trigger for GITScm polling
- ☐ Poll SCM

**Build Environment**

☐ Trigger builds remotely (e.g., from scripts)  
☐ Build after other projects are built  
☐ Build periodically  
☐ Build when another project is promoted  
☐ GitHub hook trigger for GITSCM polling  
☐ Poll SCM

**Build Environment**

☒ Delete workspace before build starts  
☐ Abort the build if it's stuck  
☐ Add timestamps to the Console Output  
☐ Use secret text(s) or file(s)  
☐ With Ant

**Build**

☐ Copy artifacts from another project  
 Project name:   
 Which build:   
☐ Stable build only  
 Artifacts to copy:   
 Artifacts not to copy:   
 Target directory:   
 Parameter filters:   
☐ Flatten directories: ☐ Optional ☒ Fingerprint Artifacts

**Post-build Actions**

☐ Deploy war/ear to a container  
 WAR/EAR files:   
 Context path:   
 Containers:
 

Tomcat 8.x	Credentials: <input type="text" value="tomcat"/>	Tomcat URL: <input type="text" value="http://localhost:8083"/>
------------	--	--

☐ Deploy on failure

Summary: You learned main line CI pipeline creation.

## **Exercise 10: Copying and Moving Jobs**

### **I. Copying Jobs:**

Step 1: Click on the option of New Item from the left panel.

Step 2: Name the new job and enter the name of the job you wish to copy below as shown:

if you want to create a new item from other existing, you can use this option:

Step 3: Click on OK and observe the newly copied job.

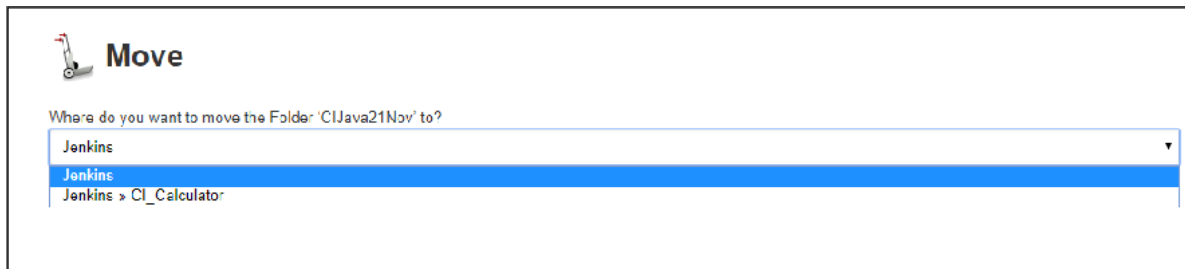
### **II. Moving Jobs:**

Step 1: Enter the folder which has the jobs that need to be moved to a new location.



Step 2: Click on the option of Move from the left panel.

Step 3: Enter the location where you wish to move the jobs to, as shown below.



Step 4: Click on Move and observe the newly moved jobs in that location.

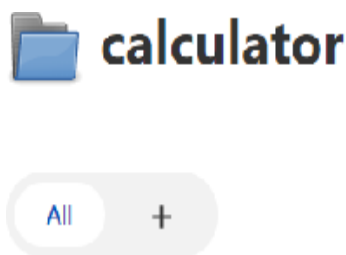
Estimated Time: 20 mins

Summary: You learned how to copy and move jobs on Jenkins.

## **Exercise 11: Creating pipeline view in Jenkins**

Objective: Understand creation of pipeline view in Jenkins

Step 1: Click on the '+' symbol under the Jenkins project folder as shown in the screenshot given below



Step 2: Provide the name for the pipeline in the viewname field and select the Build Pipeline View and click create as shown in the screenshot given below.

**View name**

Calc Pipeline View

**Type**

☒ **Build Pipeline View**  
Shows the jobs in a build pipeline view

☐ **Include a global view**  
Shows the content of a global view.

☐ **List View**  
Shows items in a simple list format. You

Create

Step 3: Select calculator->Setup as the Select Initial Job under the Upstream/downstream config as shown in the screenshot given below and click ok.

**Pipeline Flow**

Layout

Used on upstream/downstream relationship

This layout mode derives the pipeline structure based on the UI

**Upstream / downstream config**

Select Initial Job ?

calculator -> Setup

**Step 4:** Execute the job from setup and you can see the pipeline flow as shown in the screenshot given below.



**Summary:** You learned to create pipeline view in Jenkins.

## Exercise 12: Configuring Gating Conditions

**Objective:** Configure gating conditions in Jenkins.

Apply gates in the tool

Gating condition in Static Analysis job:

Step 1: Start "SonarQube" server if not started earlier.

Step 2: To create quality gate in SonarQube log into SonarQube (<http://localhost:9000>) as admin (username and password is admin) ->

**Quality Gates** -> **Create** -> enter "CALQG" as name -> enter "Critical Issues" as metric in "add condition" drop down list and create a rule as shown below and click on add button.

Complexity	Value	is greater than	100	360	Update	Delete
------------	-------	-----------------	-----	-----	--------	--------

Similarly, add another four conditions as shown below

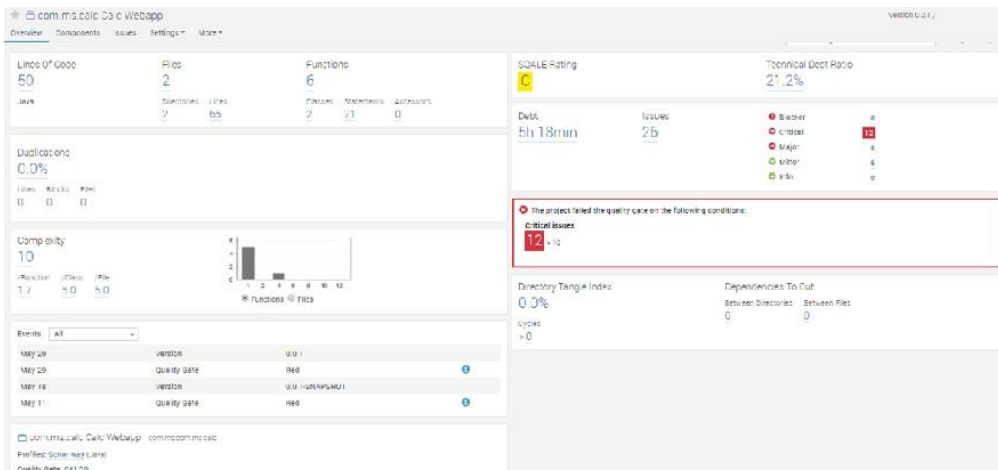
Critical issues	Value	is greater than	10	30	Update	Delete
Duplicated lines	Value	is greater than	50	55	Update	Delete
Major issues	Value	is greater than	10	15	Update	Delete
SQ/LE Technical Debt Ratio	Value	is greater than	1.2	1.4	Update	Delete

**Step 3:** To link the "CALQG" with project, use below shown option available in the same page (check the project name in without section, then that gate applied)

**PROJECTS**

With	Without	All
<input type="checkbox"/> com.ms.calc Calc Webapp		

Once static analysis done (by running Maven Target) using SonarQube, you can observe the current analysis results on SonarQube dashboard as shown below.



**Step 4:** Enabling build breaker in SonarQube, go to settings->build breaker-> set default to build breaker skip on alert flag option, go to settings -> general settings-> enter value “buildBreaker” in the text field Plugins accepted for Preview and Incremental modes.

Note: this plugin causes Maven target failure, because of quality gate violation

**Step 5:** Choose the first option as shown below in Jenkins job configuration to link with downstream job in the pipeline.

Note: You can configure default quality gates also in SonarQube.

## Gating in Jenkins

### Gating condition in Unit Testing job:

**Step 1:** You can use unit test results to decide the stability of build by setting up health report amplification factor as shown below.

**Step 2:** Choose the first option as shown below in Jenkins job configuration to link with downstream job in the pipeline.

**Step 3:** Go to the project in Eclipse and edit the junit test class (CalculatorTest.java file under src/java/test/ut)

**Step 4:** Modify any one of the testcases to get fail as shown in the screenshot given below.

```

@Test
public void testAdd() {
    assertTrue(cl.doAdd(1, 2) == 4);
}

```

**Step 5:** Commit the code and check the Jenkins pipeline for the failure.

**Step 6:** You can also fail few more test cases and observe the result in the Jenkins pipeline.

### Gating condition in Code Coverage job:

#### Method 1:

**Step 1:** You can use code coverage results to decide the stability of build as shown below.

**Record JaCoCo coverage report**

Path to exec files (e.g.: \*\*/target/\*\*.exec):  Inclusions (e.g.: \*\*/\*class):  Exclusions (e.g.: \*\*/\*Test\*.class):

Path to class directories (e.g.: \*\*/target/classDir, \*\*/classes):

Path to source directories (e.g.: \*\*/mySourceFiles):  Inclusions (e.g.: \*\*/\*java,\*\*/\*groovy,\*\*/\*.gs):  Exclusions (e.g.: generated/\*\*/\*java):

☐ Disable display of source files for coverage

☒ Change build status according to the thresholds

☐ Always run coverage collection, even if build is FAILED or ABORTED

	Instruction	% Branch	% Complexity	% Line	% Method	% Class
🔔	<input type="text" value="0"/>	<input type="text" value="90"/>	<input type="text" value="0"/>	<input type="text" value="90"/>	<input type="text" value="90"/>	<input type="text" value="0"/>
🔕	<input type="text" value="0"/>	<input type="text" value="0"/>	<input type="text" value="0"/>	<input type="text" value="0"/>	<input type="text" value="0"/>	<input type="text" value="0"/>

☒ Fail the build if coverage degrades more than the delta thresholds

	Instruction	% Branch	% Complexity	% Line	% Method	% Class
🔕	<input type="text" value="0"/>	<input type="text" value="70"/>	<input type="text" value="0"/>	<input type="text" value="80"/>	<input type="text" value="80"/>	<input type="text" value="0"/>

Add post-build action

**Step 2:** Choose the first option as shown below in Jenkins job configuration to link with downstream job in the pipeline.

**Build other projects**

Projects to build:

☒ Trigger only if build is stable

☐ Trigger even if the build is unstable

☐ Trigger even if the build fails

Delete

Here is a summary of the activities to be done

- Open SonarQube (<http://localhost:9000>). Add gating conditions as learnt in the demos. Give values so that you can observe a broken and a smooth build.
- Trigger a build in Jenkins and observe the gating condition in SonarQube working.

## DevOps-Continuous Integration and Continuous Delivery

c) Trigger a build in Jenkins and observe the gating condition in JaCoCo working.

d) Configure the threshold values for unit testing in Jenkins. Make changes to the unit tests (so that some of them fail) and observe the gating conditions failing in Jenkins. Correct them and observe that the build becomes stable. (when changes are being made, commit the test code from Eclipse).

Summary of this Exercise:

You have learnt apply gating conditions to ensure code quality and tests pass in every build.

## **Additional Exercises**

### **1. Adding custom rules to SonarQube**

**Objective:** Add custom rules to Sonarqube

**Requirements:** SonarQube 5.6 and above

Step 1: Run the StartSonar.bat  StartSonar f file as admin.

#### **Creating the custom rule**

##### **Rule to be created:**

Avoid single parameter in methods. Here are the parameters:

1. name: Avoid usage of single parameter
2. Description: This rule expects methods to have at least two parameters
3. This is a coding guideline
4. Priority is MAJOR

Add this rule to Sonarqube and check the same using the calculator workspace provided

**Hint:** Use method.parameters().size() to check the number of parameters in a method Estimated time: 30 mins

**Summary:** You have learnt to customize the rules in SonarQube according to your requirement for quality analysis.

### **2. Static program analysis using SonarLint**

**Objective:** Perform static program analysis during coding using SonarLint plug-in to Eclipse IDE.

SonarLint is a plug-in to an IDE that provides on the fly feedback to developers on new bugs and quality issues injected into their code.

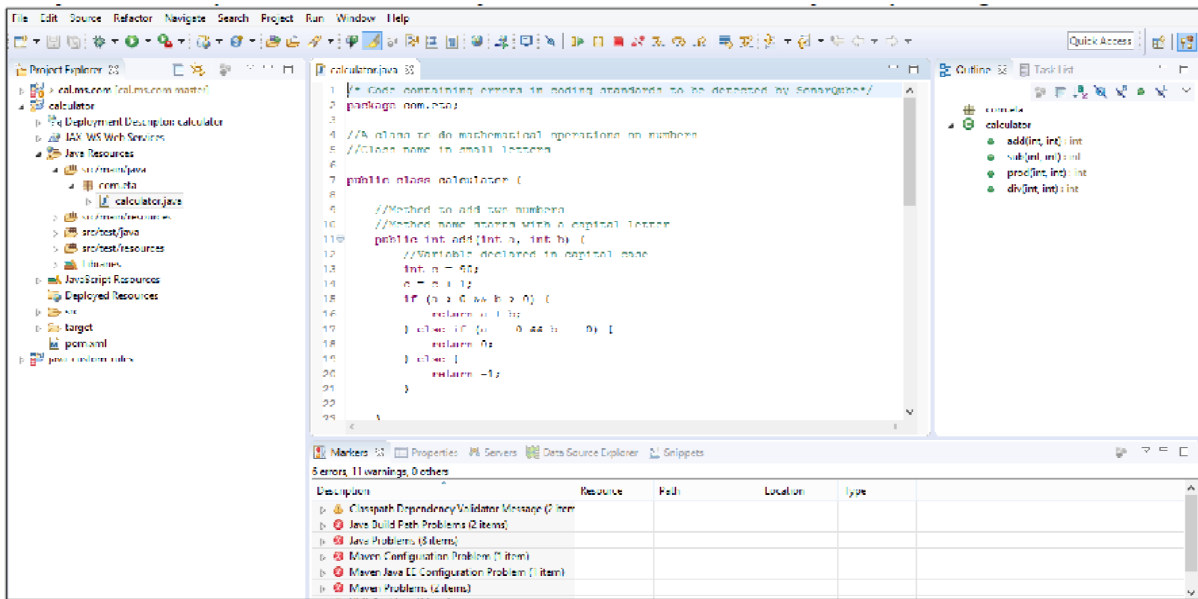
#### **Installing the SonarLint plug-in**

Step 1: Go to Eclipse IDE and select a workspace.

Step 2: Import the project **JNTU\_Calc\_Application** provided and switch to

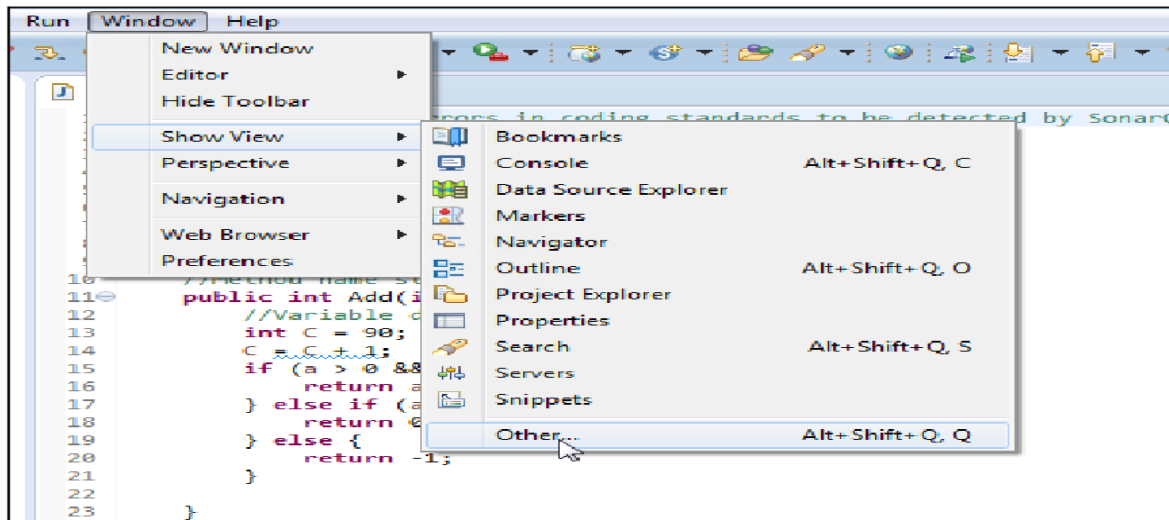
JAVA EE perspective.

Step 3: Now open the calculator.java file from 'src/main/java' package

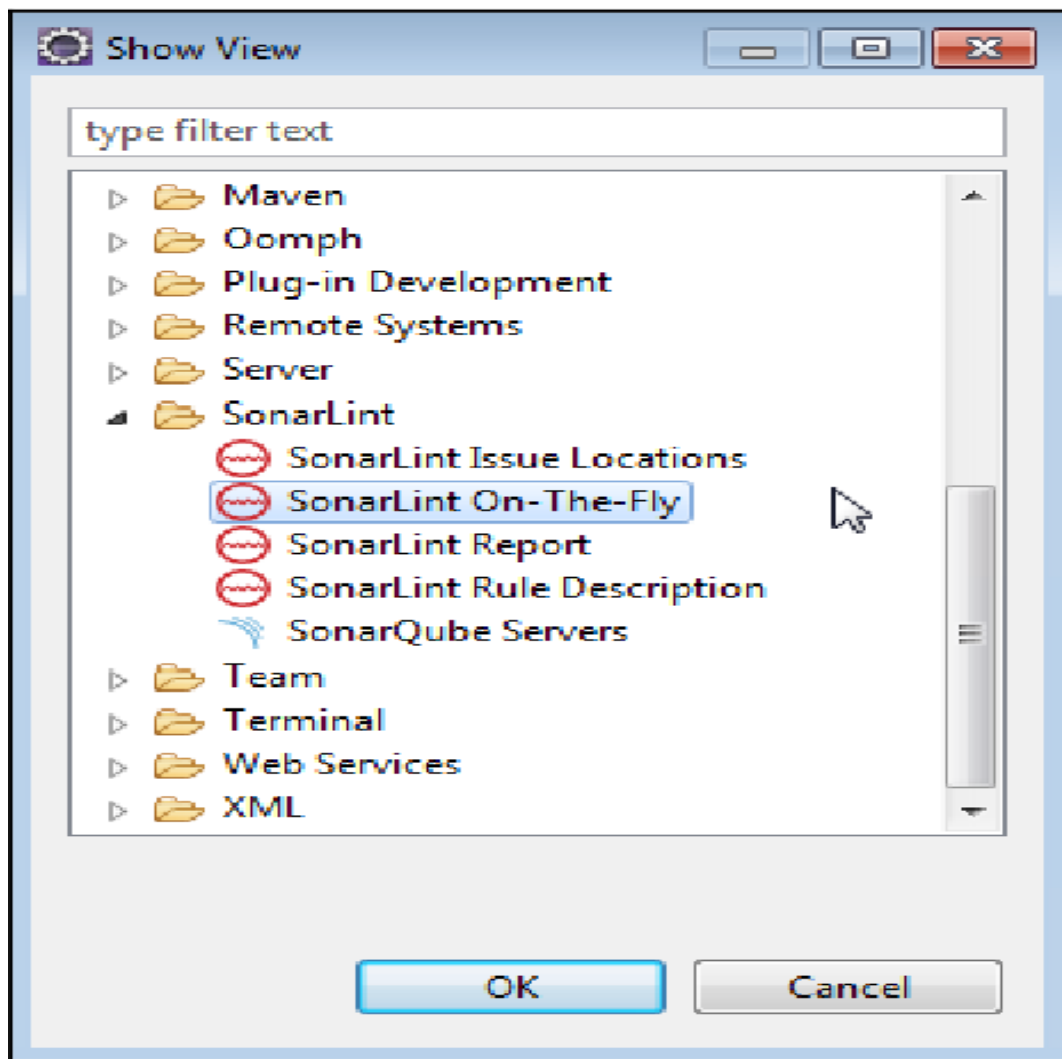


**Working with SonarLint:**

**Step 1:** Go to Windows>Show View>Other.



re select SonarLint>SonarLint On-The-fly.

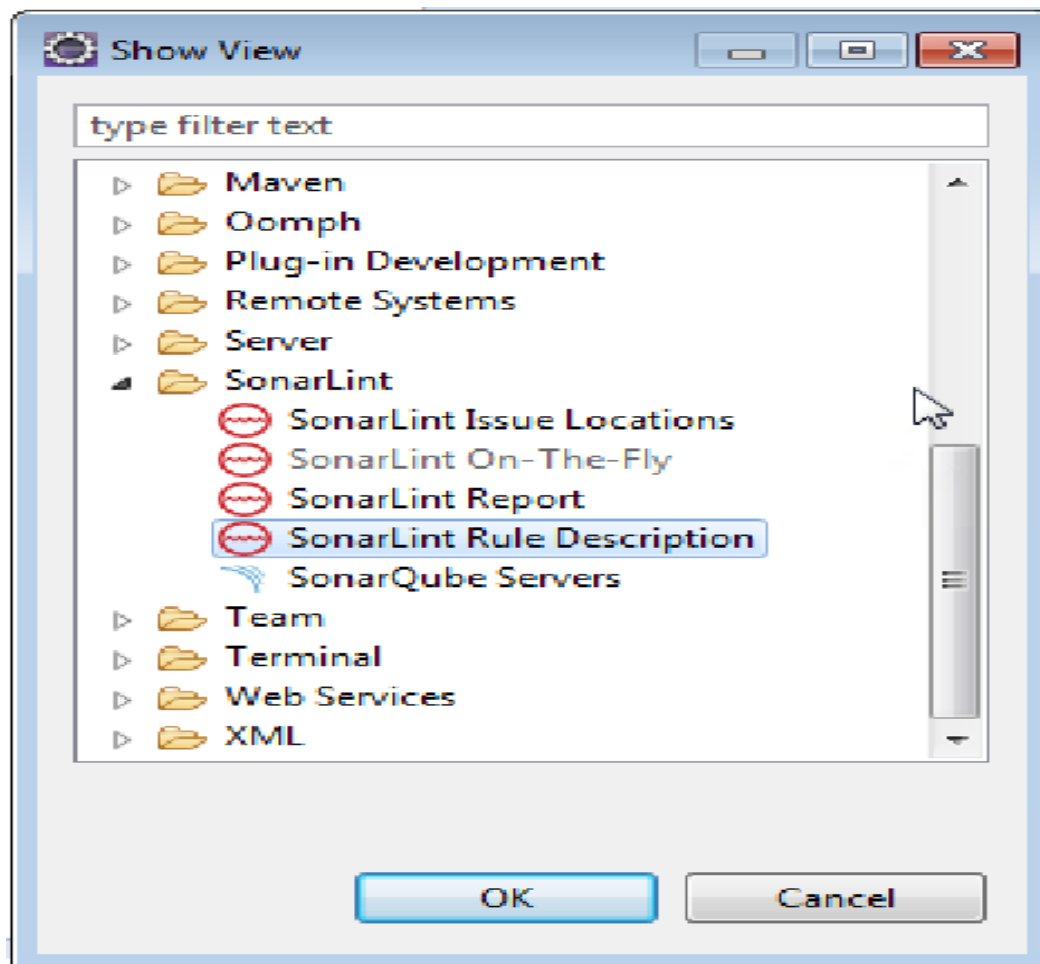


A new tab opens below which shows the issues found in the current java file selected.

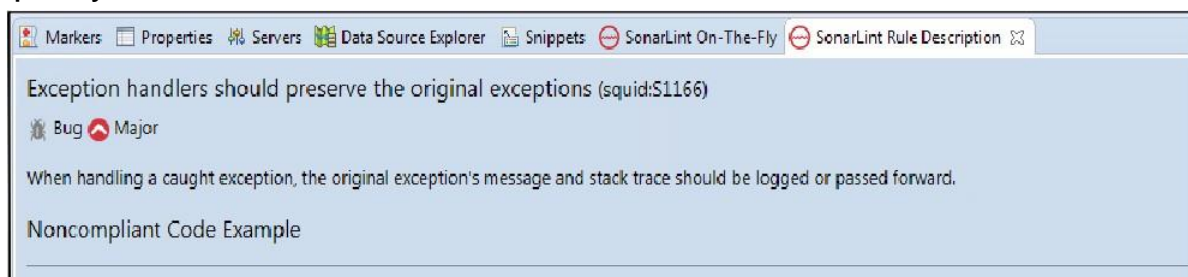
2 items			
Date	Description	Resource	
	⚠ Either log or rethrow this exception.	calculator.java	
	⚠ Remove this useless assignment to local variable "C".	calculator.java	
Issues reported "on the fly" by SonarLint on files you have recently opened/edited			

Step 2: Navigate to Windows>Show view>other. Here select SonarLint>SonarLint Rule Description.





A new empty SonarLint Rule Description tab opens. Now select one of the issue in the SonarLint On-the-fly tab and switch to SonarLint Rule Description tab. This suggests the necessary changes to be made in the code to retain the quality of the code.



Estimated time: 20 mins

Summary: You have learnt the static code analysis using SonarLint in this exercise



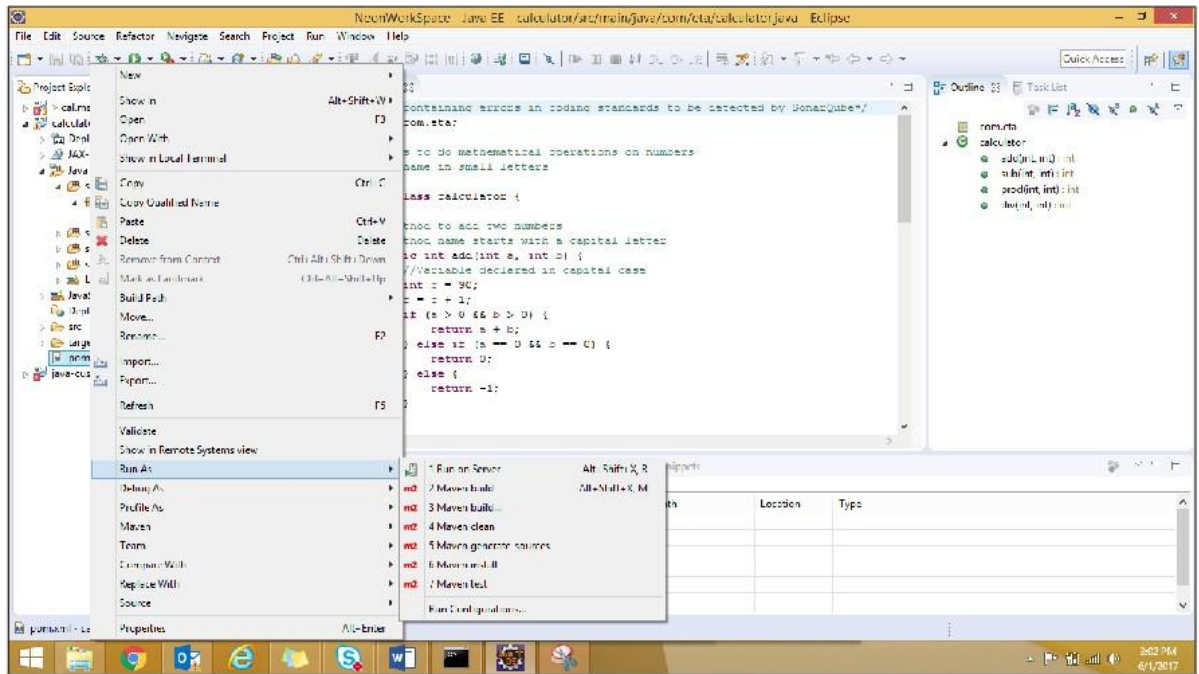
### 3. Binding SonarQube rules to SonarLint

**Objective:** Customize the rules used by SonarLint through SonarQube web server interface.

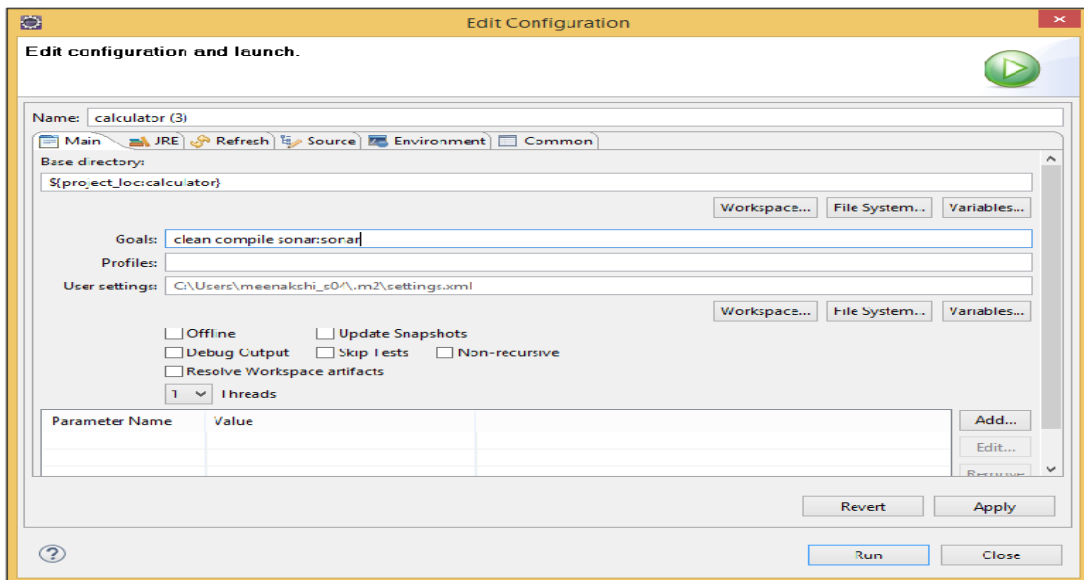
**Requirements:** SonarQube 5.6 and above

Step 1: Run the StartSonar.bat file as admin.

Step 2: In the project imported in the previous exercise, run the pom.xml as Maven build.

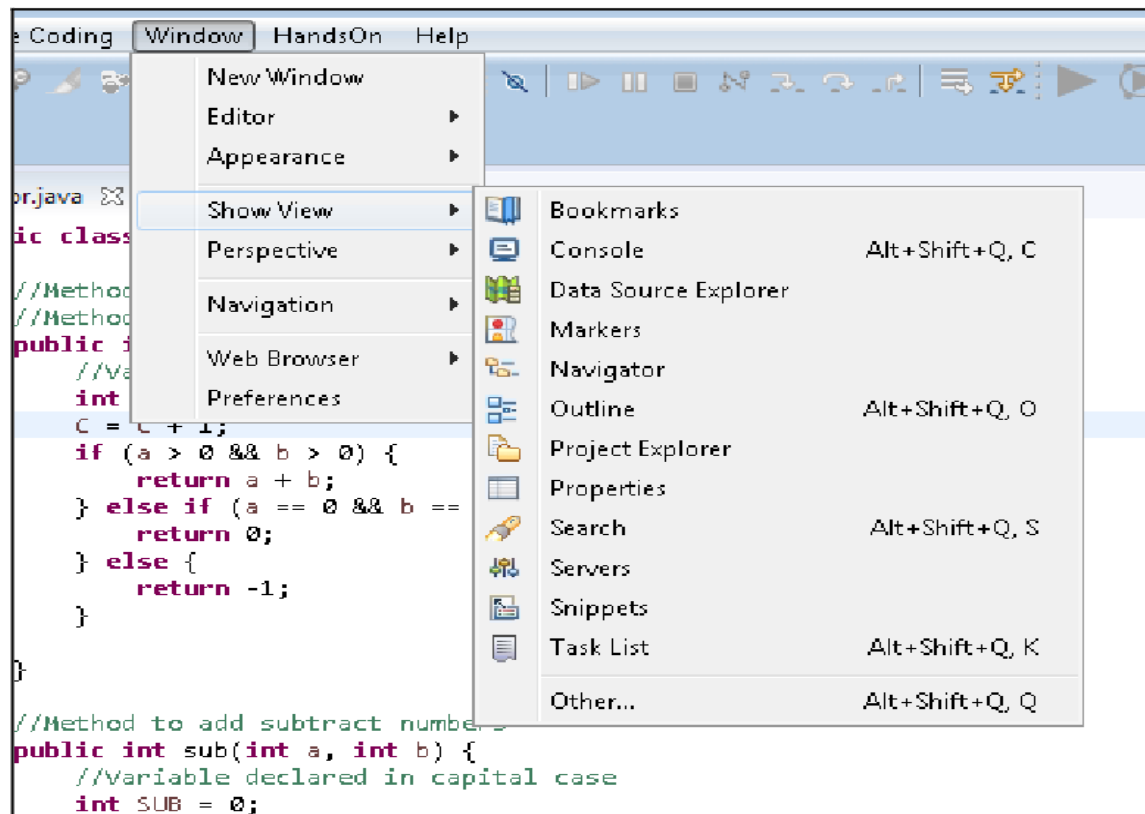


In the Goals tab, use clean, compile and sonar: sonar and execute the pom.xml file.

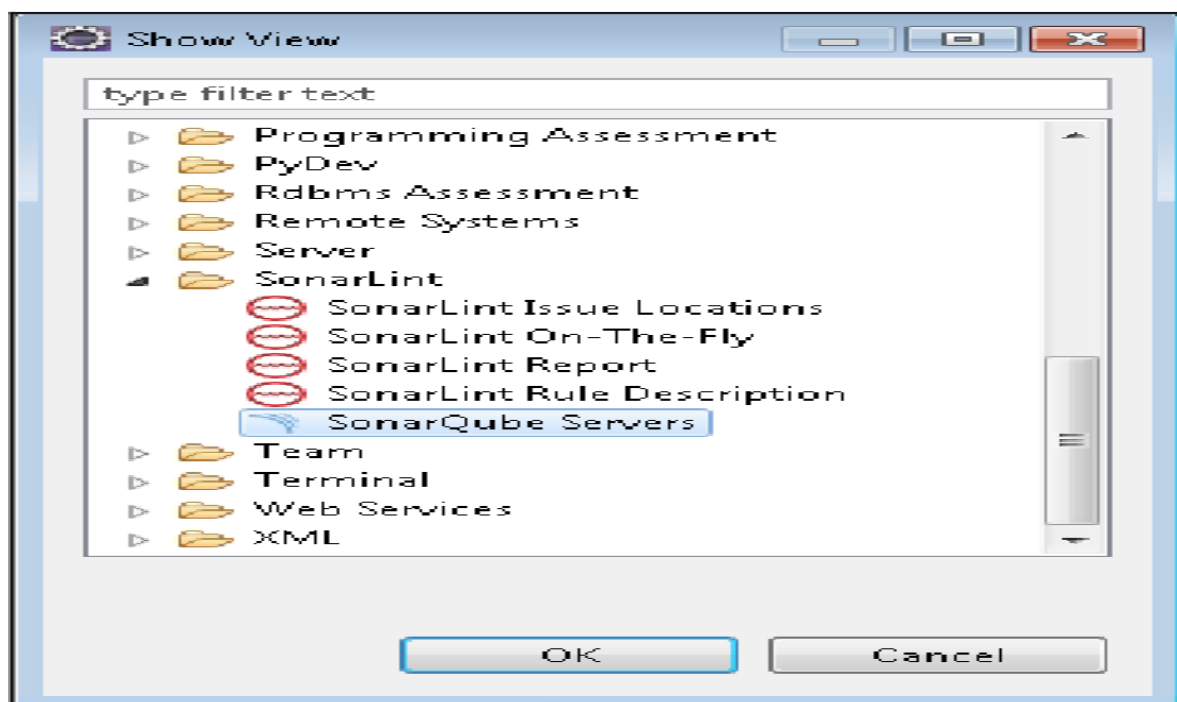


Once the build is successful, we need to bind the current java project with the SonarQube project.

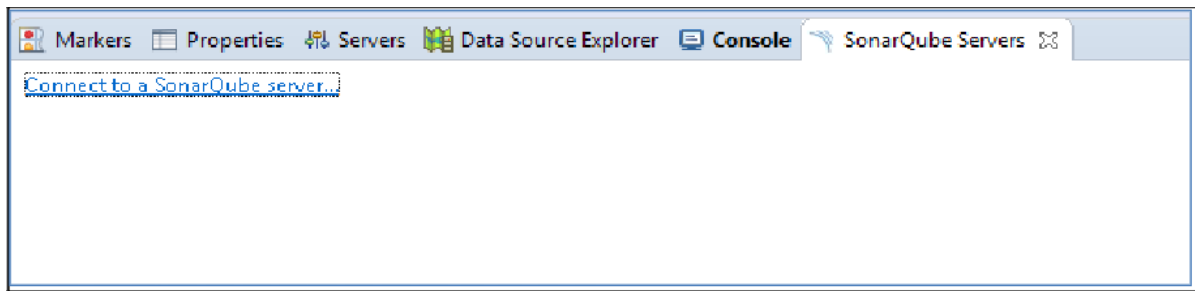
Step 3: Go to Window->Show View->Other.



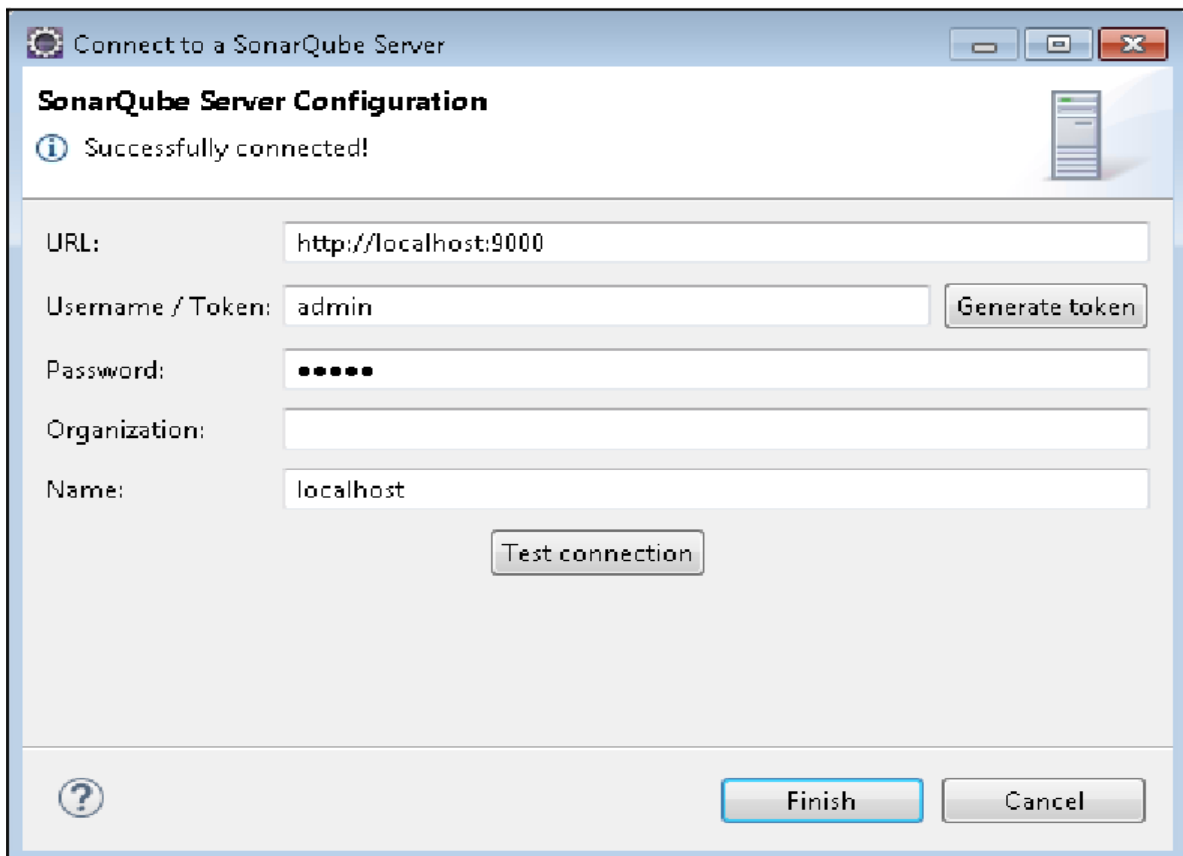
Select SonarLint->SonarQube Server.



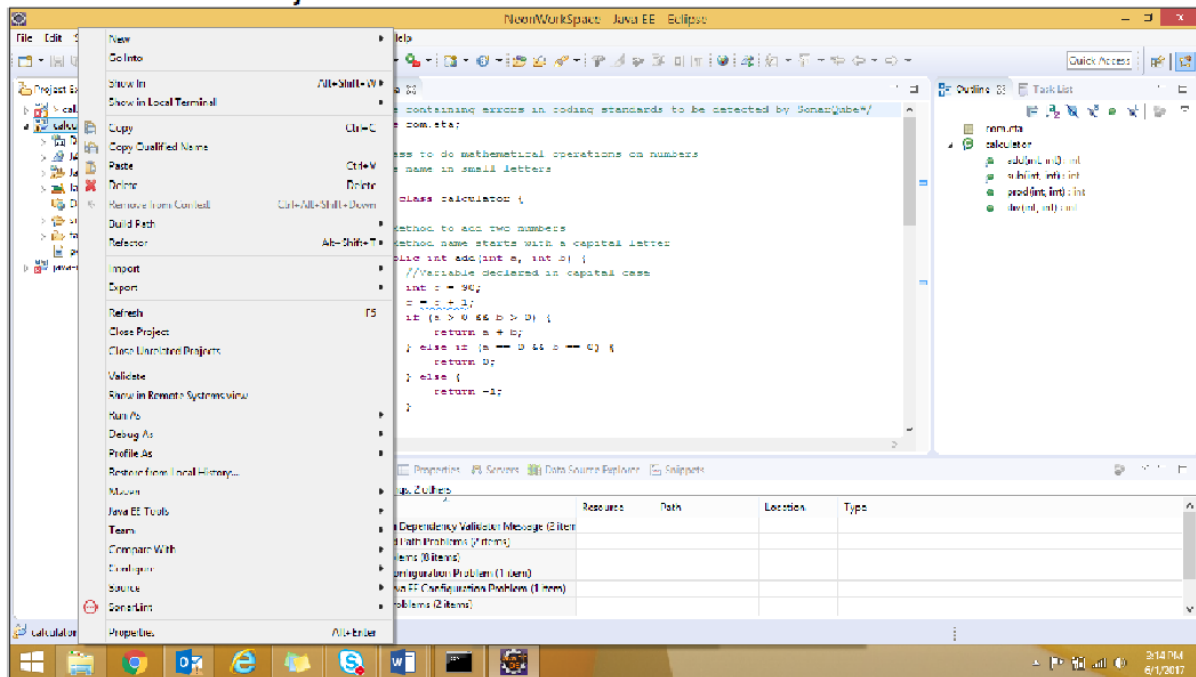
This opens a SonarQube Servers tab below. Click on the Connect to a SonarQube Server option showing up in the tab.



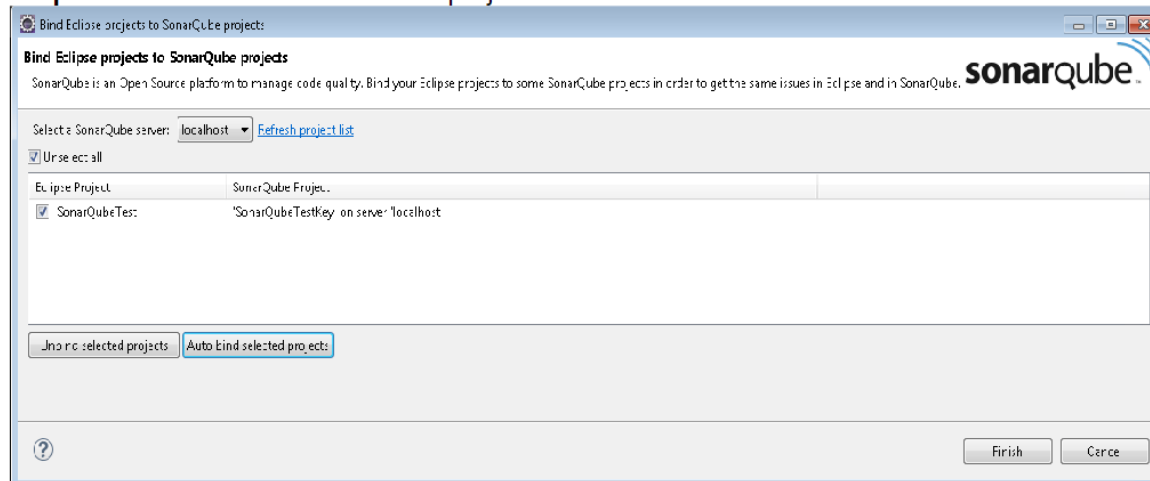
**Step 4:** Put the following details Username:admin Password:password as shown in the image given below and then click on TestConnection button.



**Step 5:** Go back to Eclipse, right click on the project under SonarLint click on 'Bind to a SonarQube Project'.



**Step 6:** Click on Auto bind selected projects and click on Finish.



Once you click on Auto bind, you will see 'SonarQubeTestKey/ on server localhost' under the title SonarQubeProject.

**Step 7:** Now open a browser and open <http://localhost:9000> to view the project on the SonarQube server.

Prepared By:  
Mr.P. Nagababu  
Assistant Professor  
CSE Dept