# Assignments for DevOps Continuous Integration and Continuous Delivery

# CONTENTS

# Assignments for DevOps Continuous Integration and Continuous Delivery

## 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 sceenshots 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:

- **JAVA_HOME** = path to jdk folder (C:\Program Files\Java)
- **M2_HOME** = path to maven folder (C:\Program Files\Maven)
- **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 below mentioned URL's and credentials

1. http://localhost:9000 – SonarQube [admin/admin]
2. http://localhost:8080 – Tomcat [tomcat/s3cret]
3. http://localhost:8064/jenkins - Jenkins

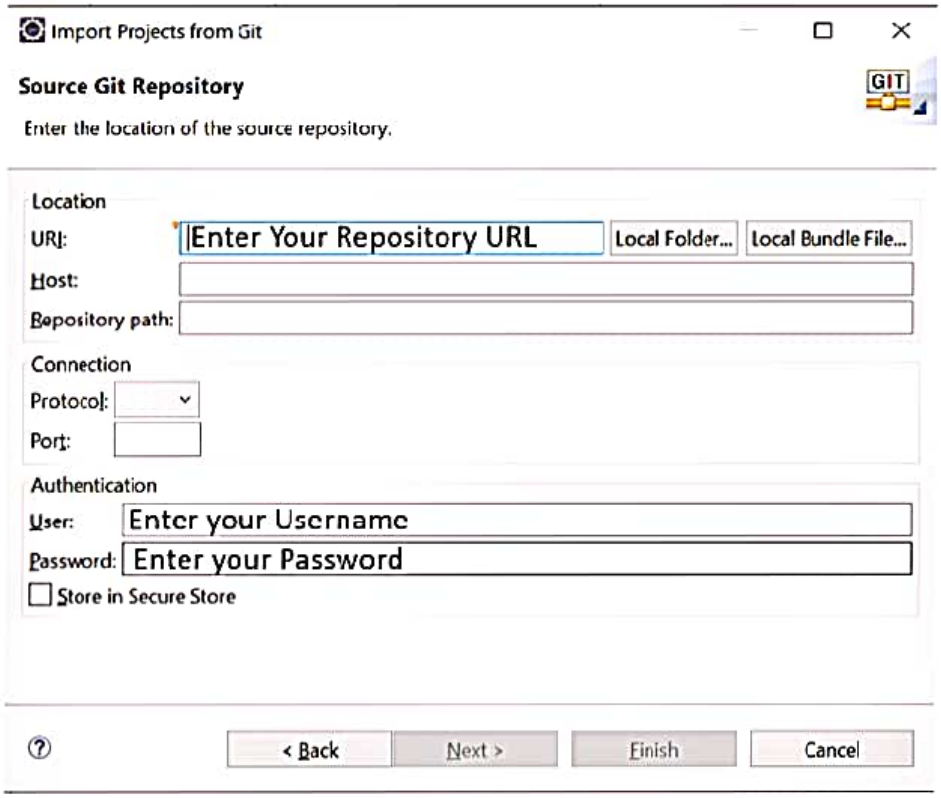
**Summary:** You have tested lab setup for the forthcoming exercises for Jenkins.

## Exercise 2: Git operations

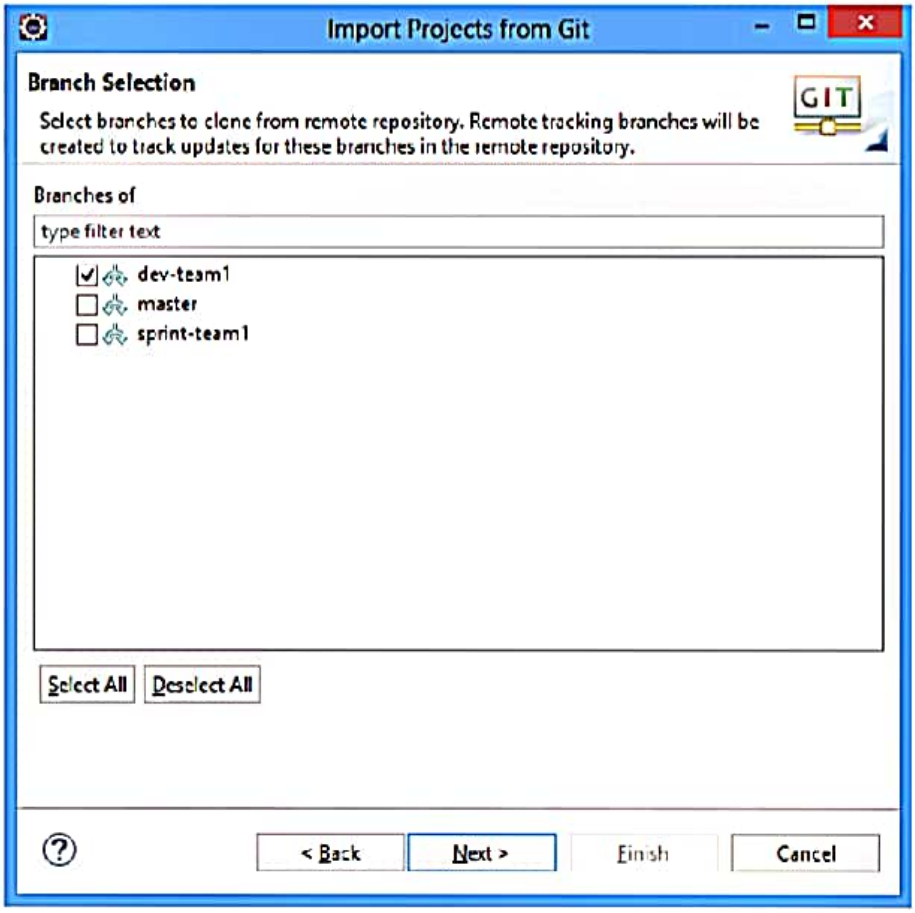
**Objective:** Perform the basic operations on Git repositories using EGit (Eclipse plugin)

**Pulling code from Git repository**

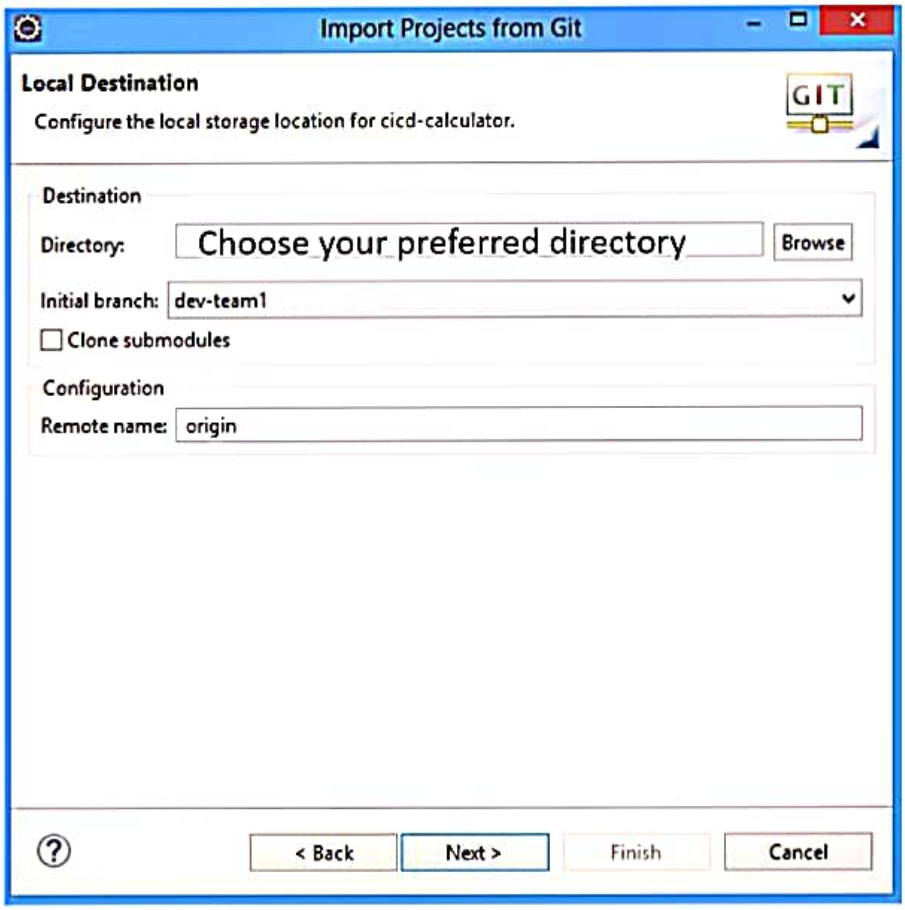
**Step 1:** Go to Eclipse-> file-> import-> Git -> Projects from Git -> Clone URI -> enter **central repository details** -> use credentials.



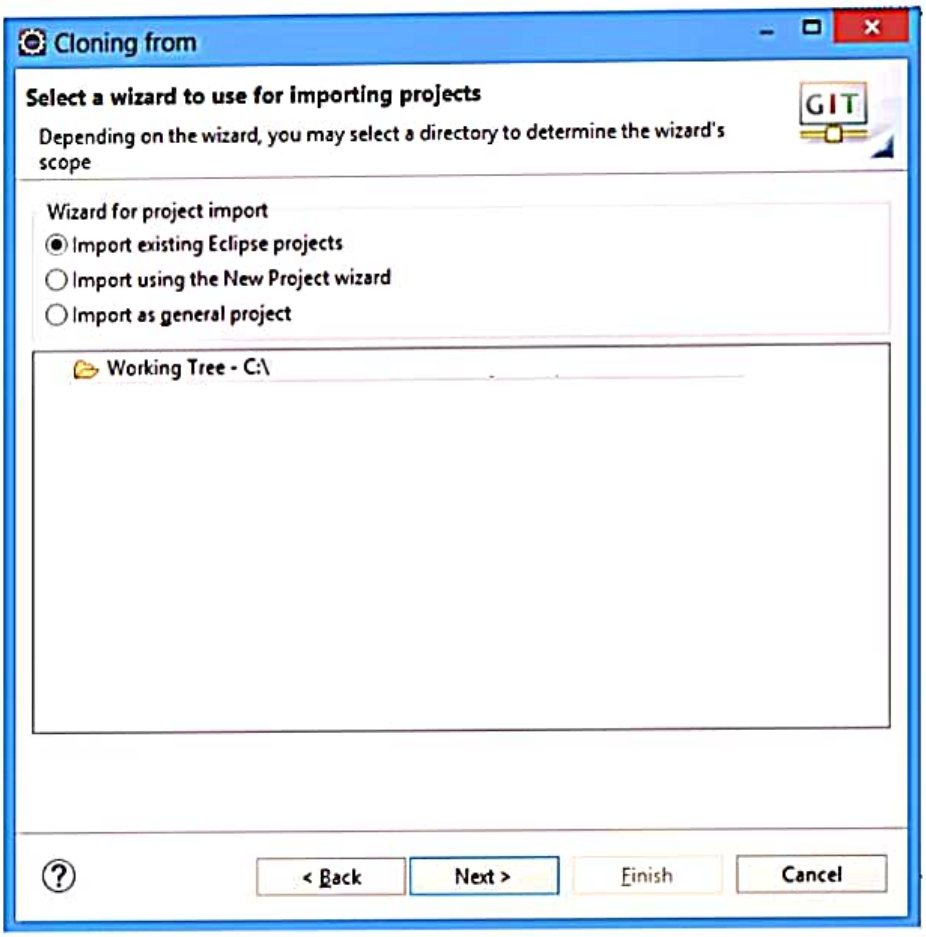

Click Next-> and choose branch (sample shown below)



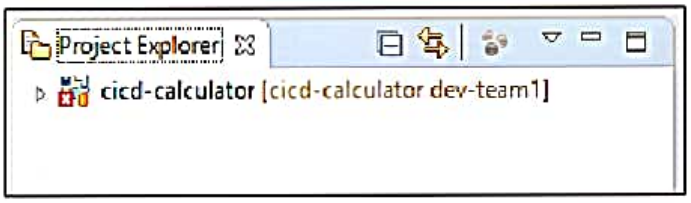

->Click next->browse directory to keep local repository as shown below->




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

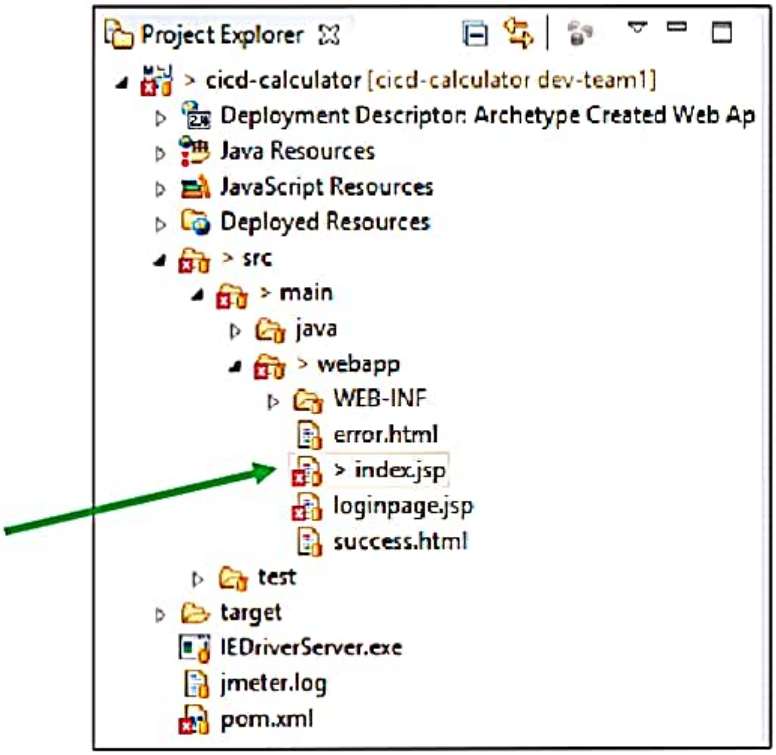



->Click Next -> click Finish->you can observe project explorer view with project cloned as shown below->. The name of the project is JNTU_Calc_Application in your case.




## 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.



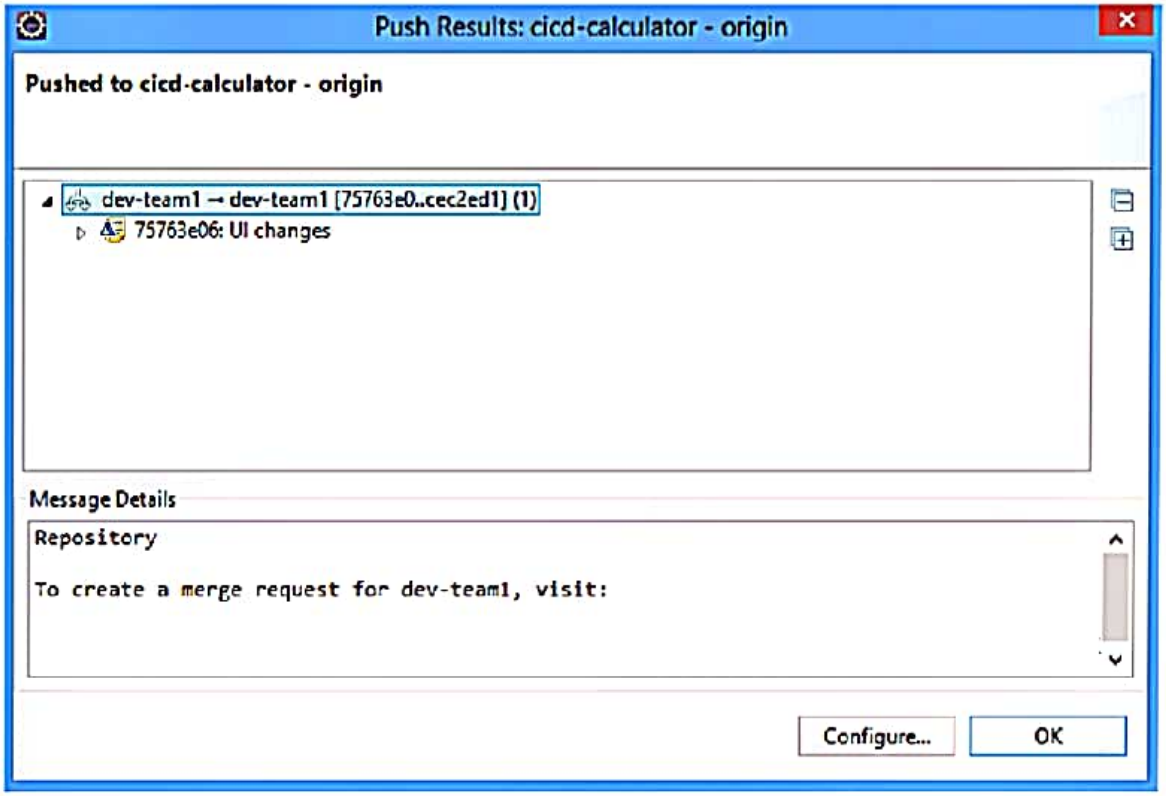

**Step 2:** Right click on project->team->commit->drag the files from Unstaged Changes to Staged Changes ->Enter appropriate comments for the commit as shown below->


->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**

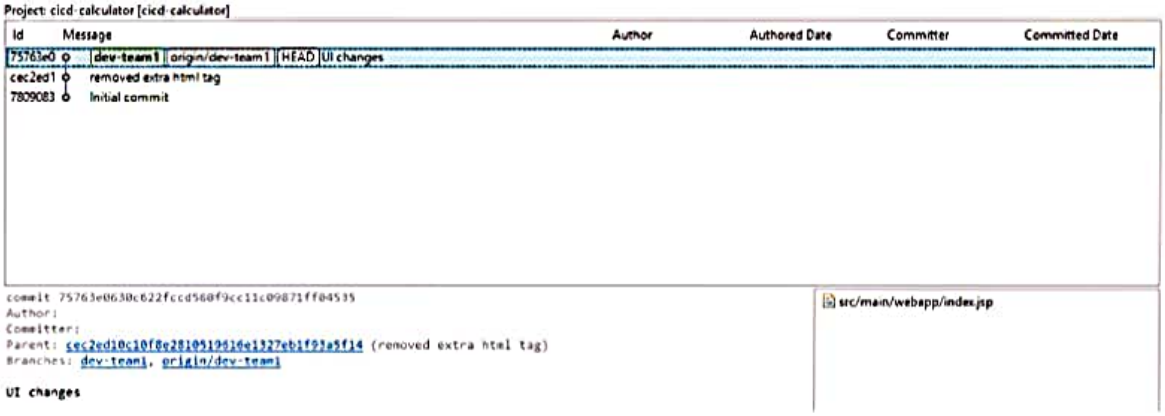
**Step 1:** Right click on project-> team-> click on Push to upstream -> you can see changes successfully pushed from local repository to central repository as shown below.
Note: Your repository name and number may be different, this is an illustration.




**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.

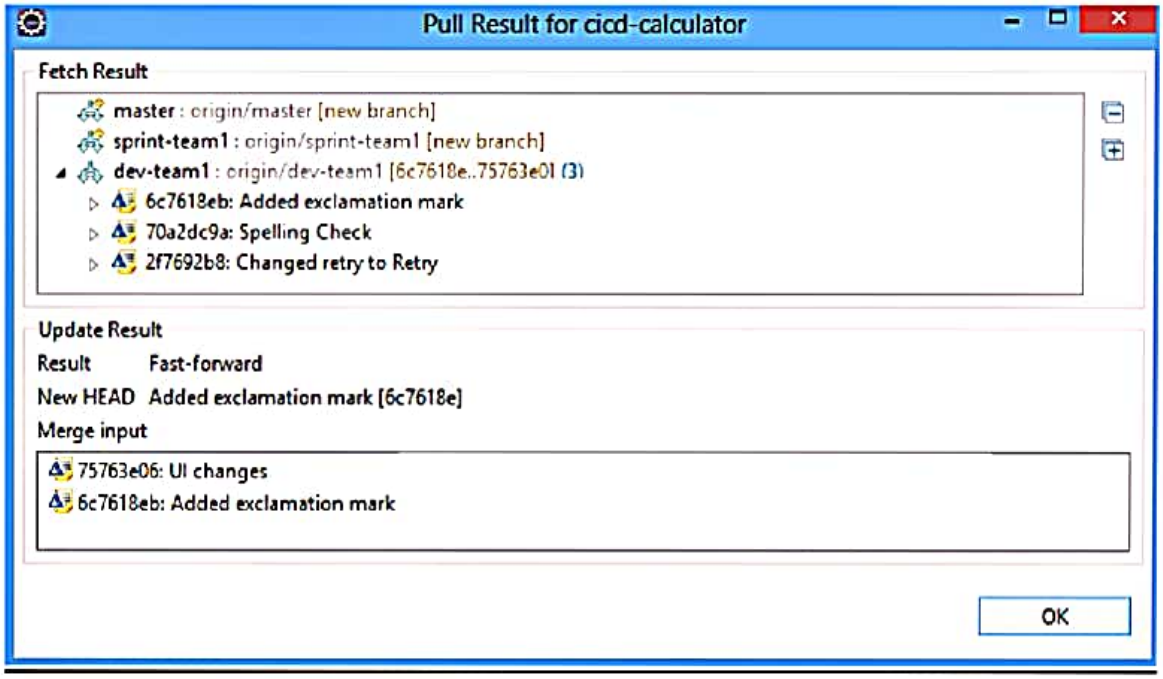
You can observe the revision history using project-> team->"Show In History" as shown below.




## 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.

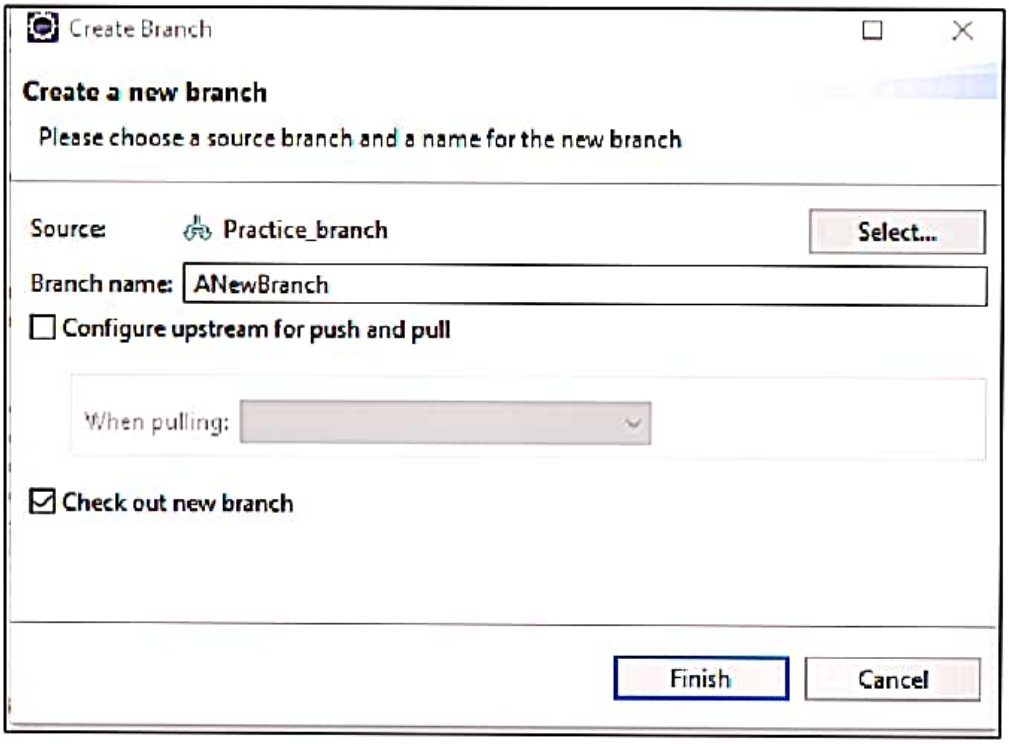
**Note:** Perform commit, and push operations on remote Git repository by making some changes to source code. You can use the repository provided in the earlier demo for completing this exercise.

**Creating branches on local repository from eclipse via e-git plug in:**

**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.**

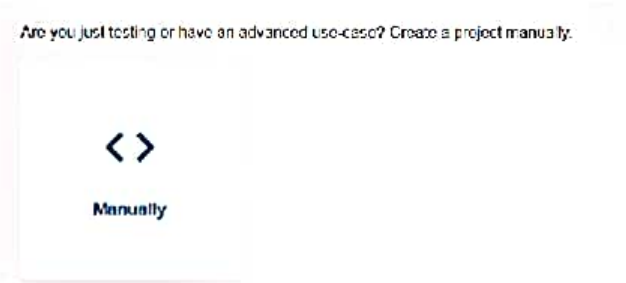


**Summary:** You have learnt to do basic operations with Git related to version control in this exercise.

## Exercise 3: Creating a project in SonarQube

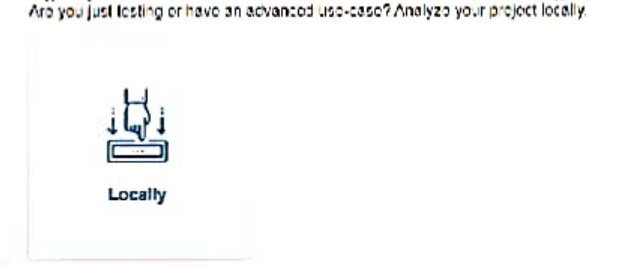
**Objective:** Understand creation of project in Sonarqube

**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.




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

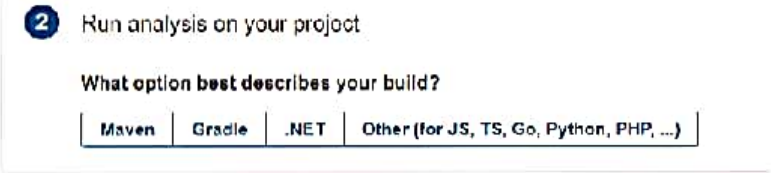

**Step 4:** Click on Locally option as shown in the screenshot given below.




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


**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.


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




**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>
```

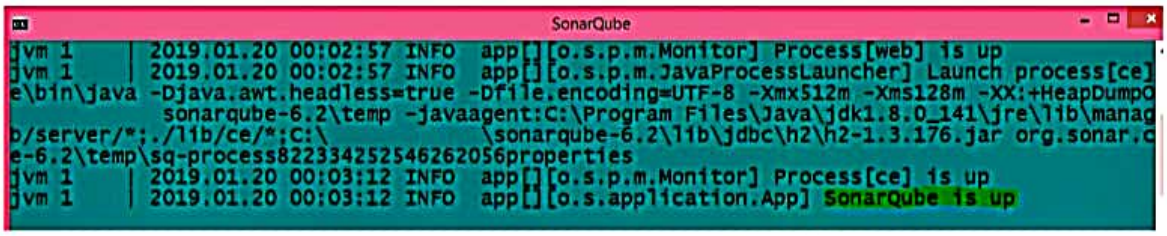
**Summary:** You have learnt to create a project in sonarqube in this exercise.

## Exercise 4: Using Sonarqube with Sonar-Runner

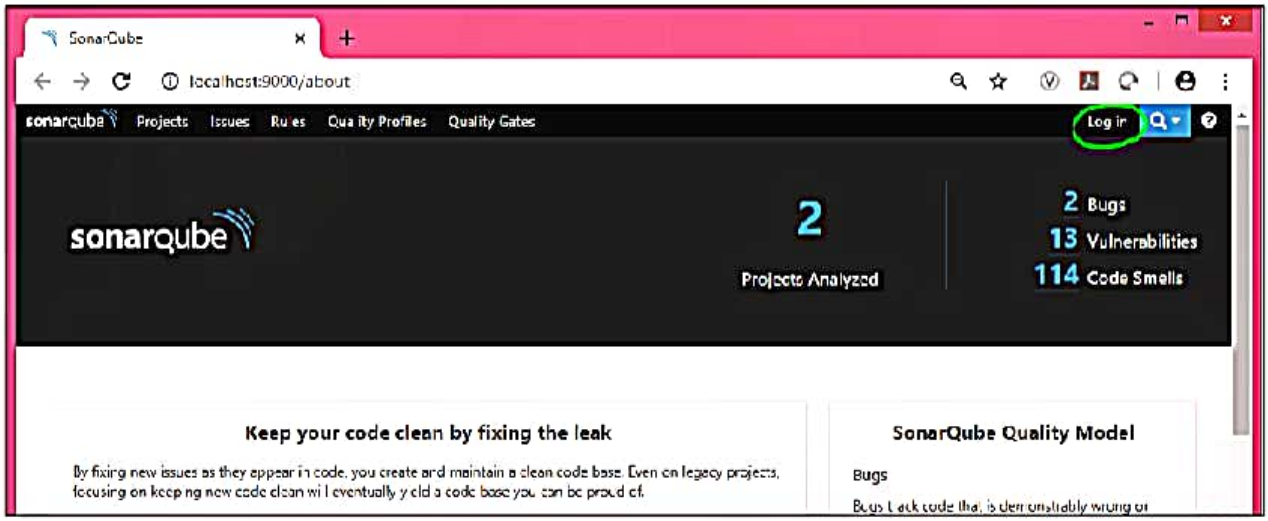
**Objective:** Understand running of Sonarqube using Sonar-Runner command-line tool

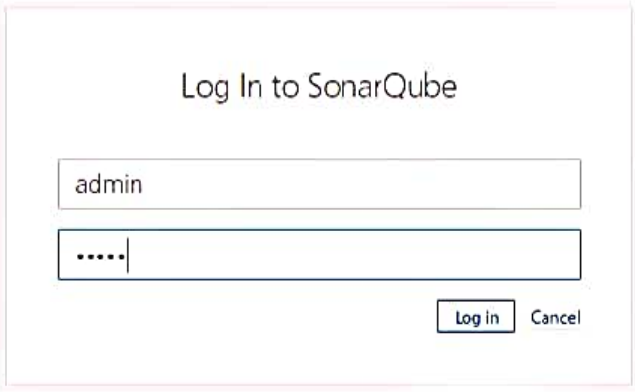
**Step 1:** Start Sonar server by using the appropriate bat file.

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

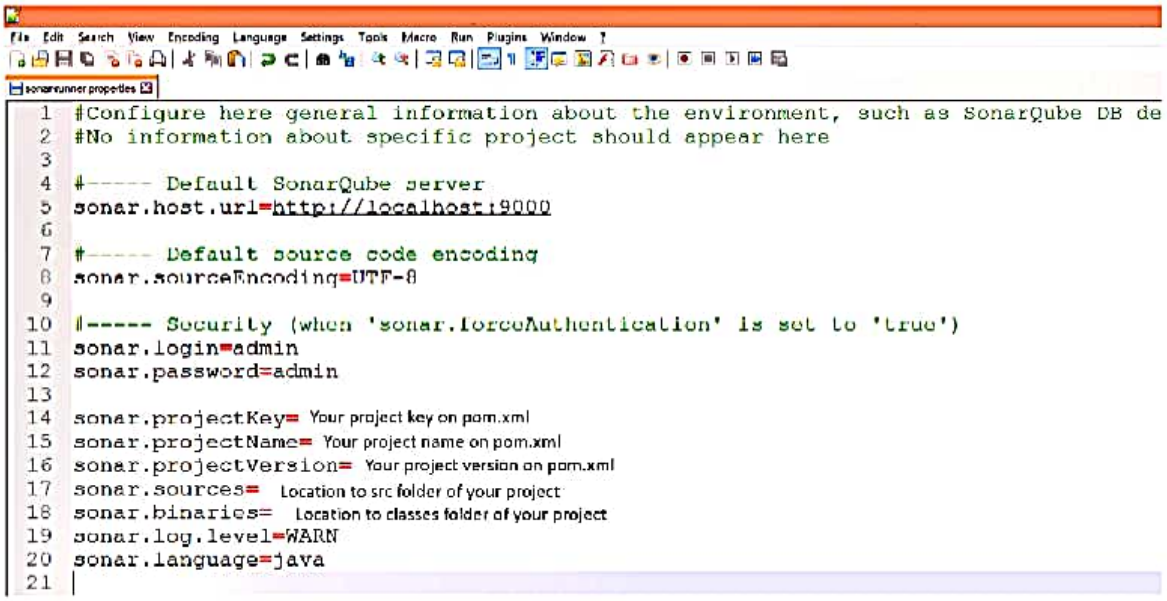



Go to SonarQube server dashboard at http://localhost:9000 and login using admin/admin

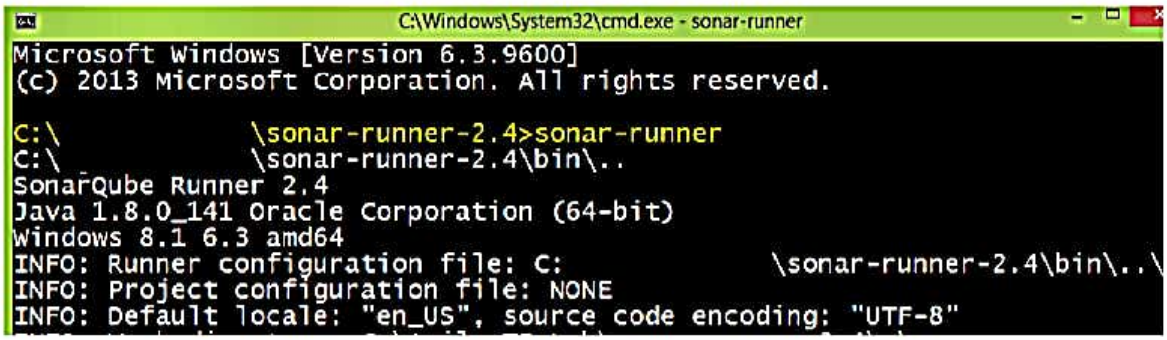



**Step 2**: Go to conf folder of Sonar Runner. Make the following changes based on the name of the Project provided to you and the path where it exists in your system:



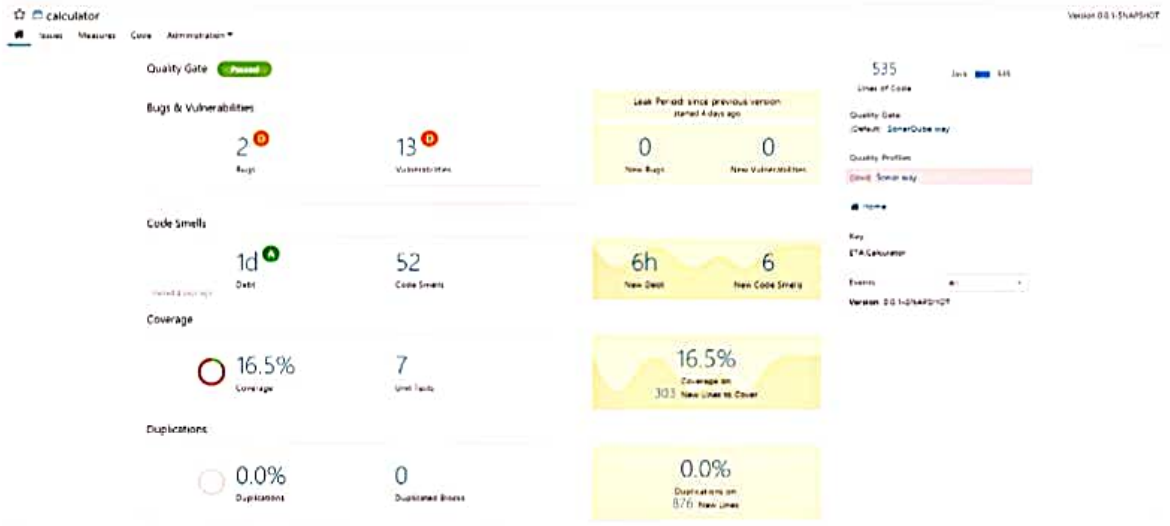

Open command prompt inside the project src folder in File Explorer and run the following command:




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>**/Calculaterut.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>
```

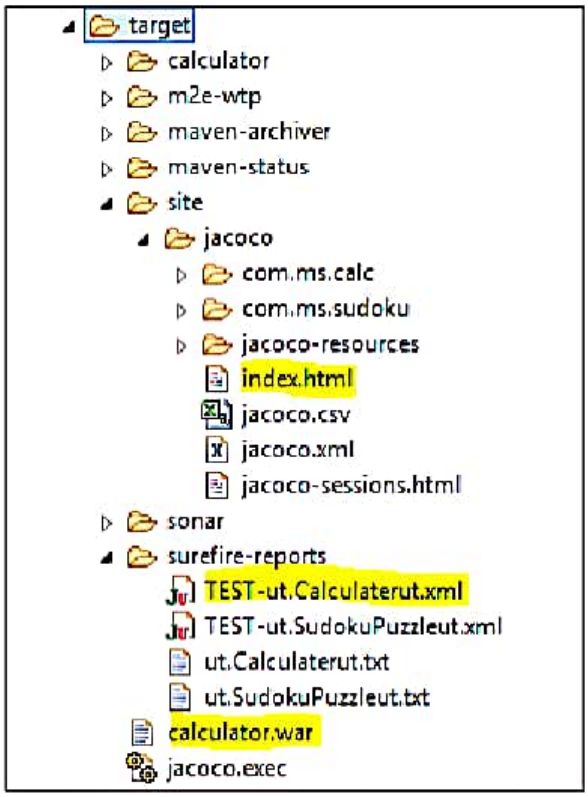
```
      <groupId>com.lazerycode.jmeter</groupId>
                                    <artifactId>jmeter-maven-
plugin</artifactId>
                                    <version>2.4.0</version>
                                <executions>
                                        <execution>
                                              <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: 43M/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: surefire-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 GlassFish or Tomcat, and then deploy Jenkins.war to it

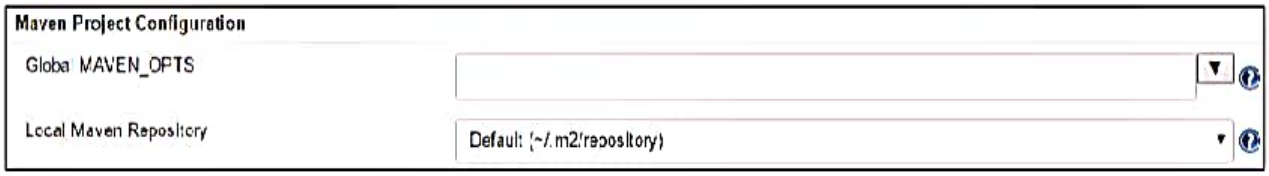
We have used the first approach in this exercise.

**Configuring Jenkins**

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

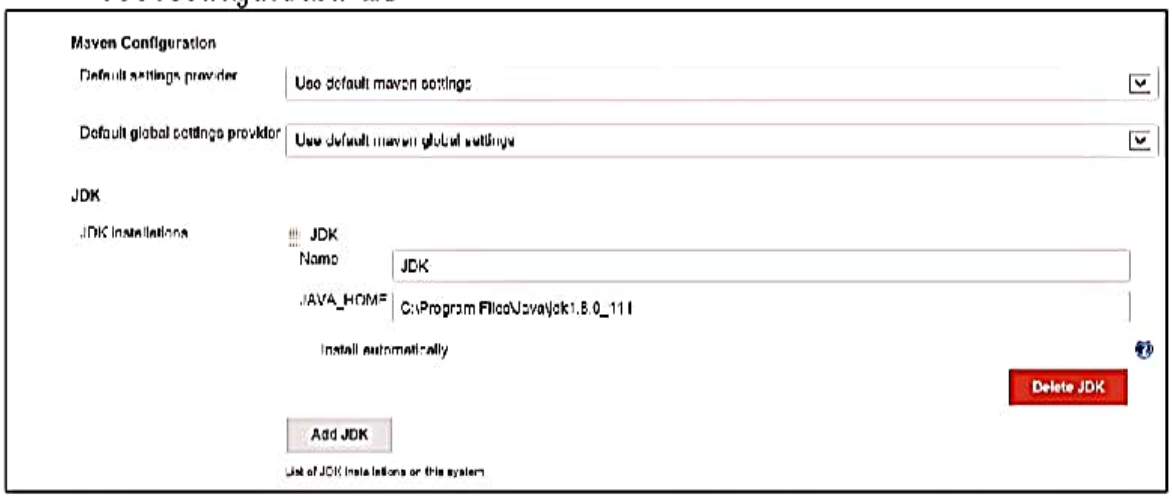
**Step 2:** Go to Manage Jenkins -> Configure System
Configure global settings and paths.

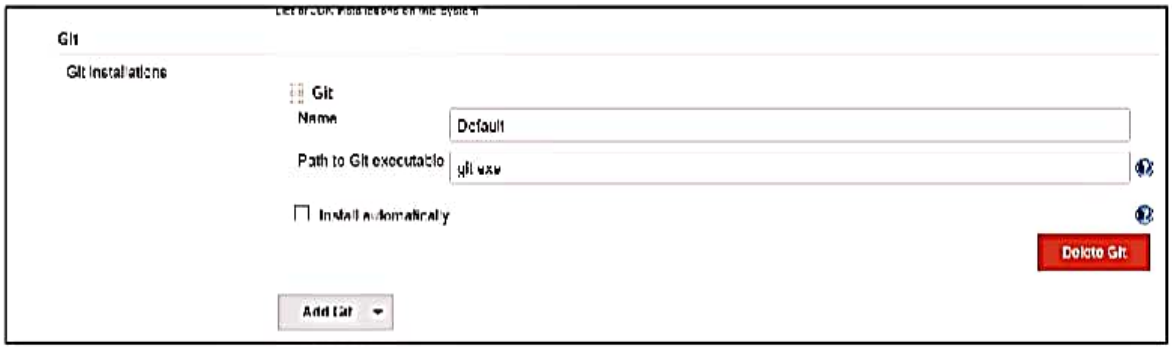
**Step 3:** You can observe Jenkins Maven integration as shown below:




## Additional configurations

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




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 artifact plugin and check the check box as shown in the screenshot given below.


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

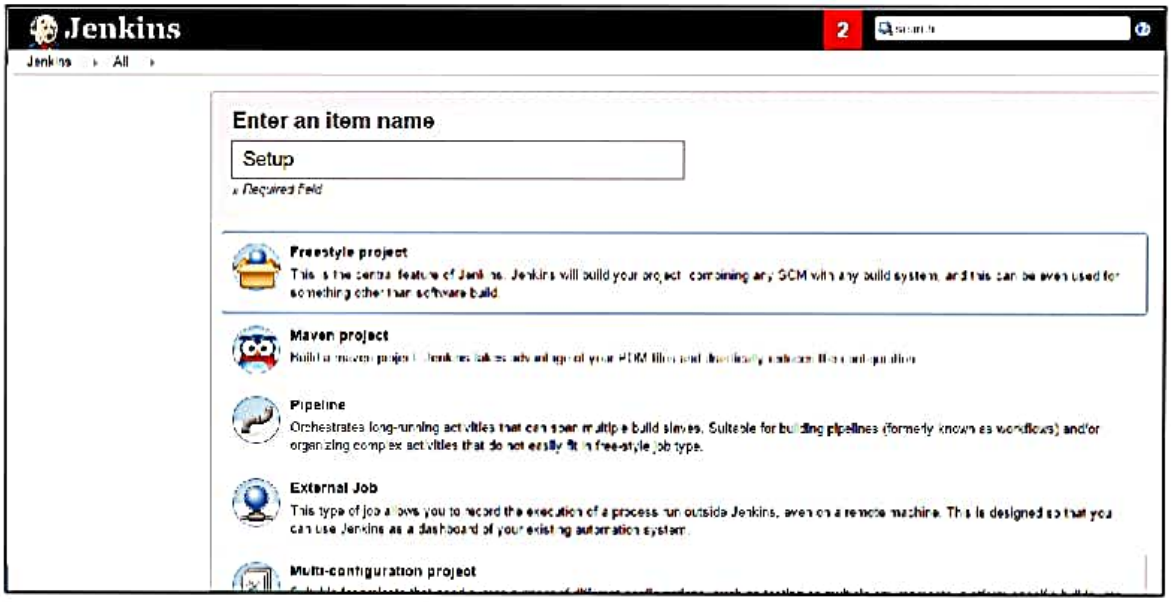
a. JaCoCo
b. Artifactory
c. Build pipeline
d. Deploy to container

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

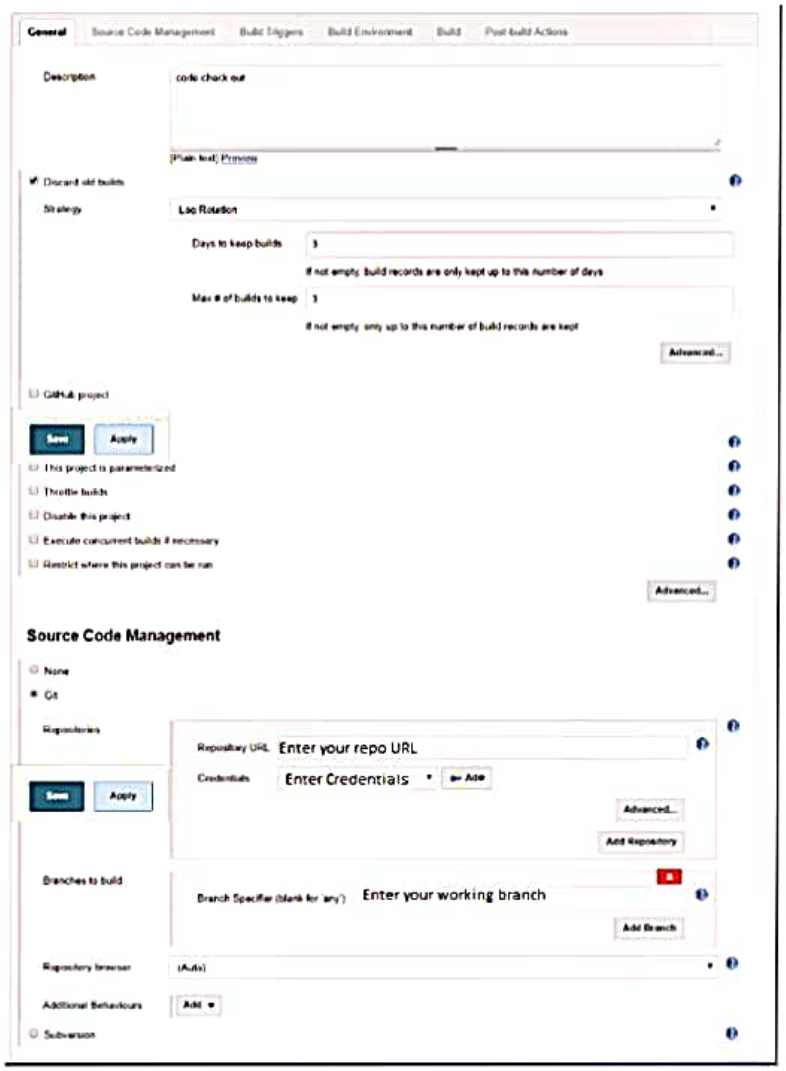
## Exercise 9: Creating Central CI pipeline

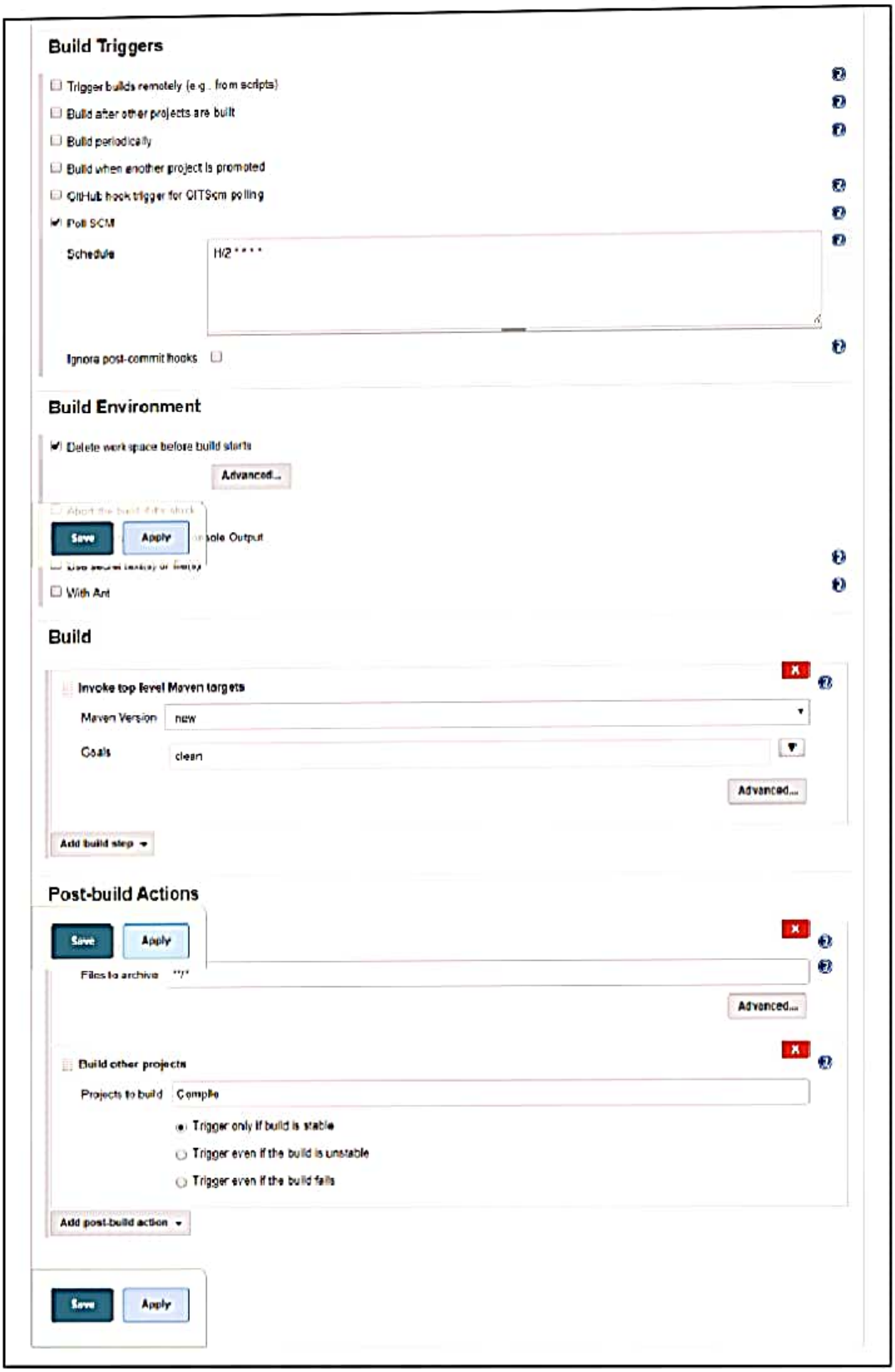
**Objective:** Creating main line CI pipeline

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.

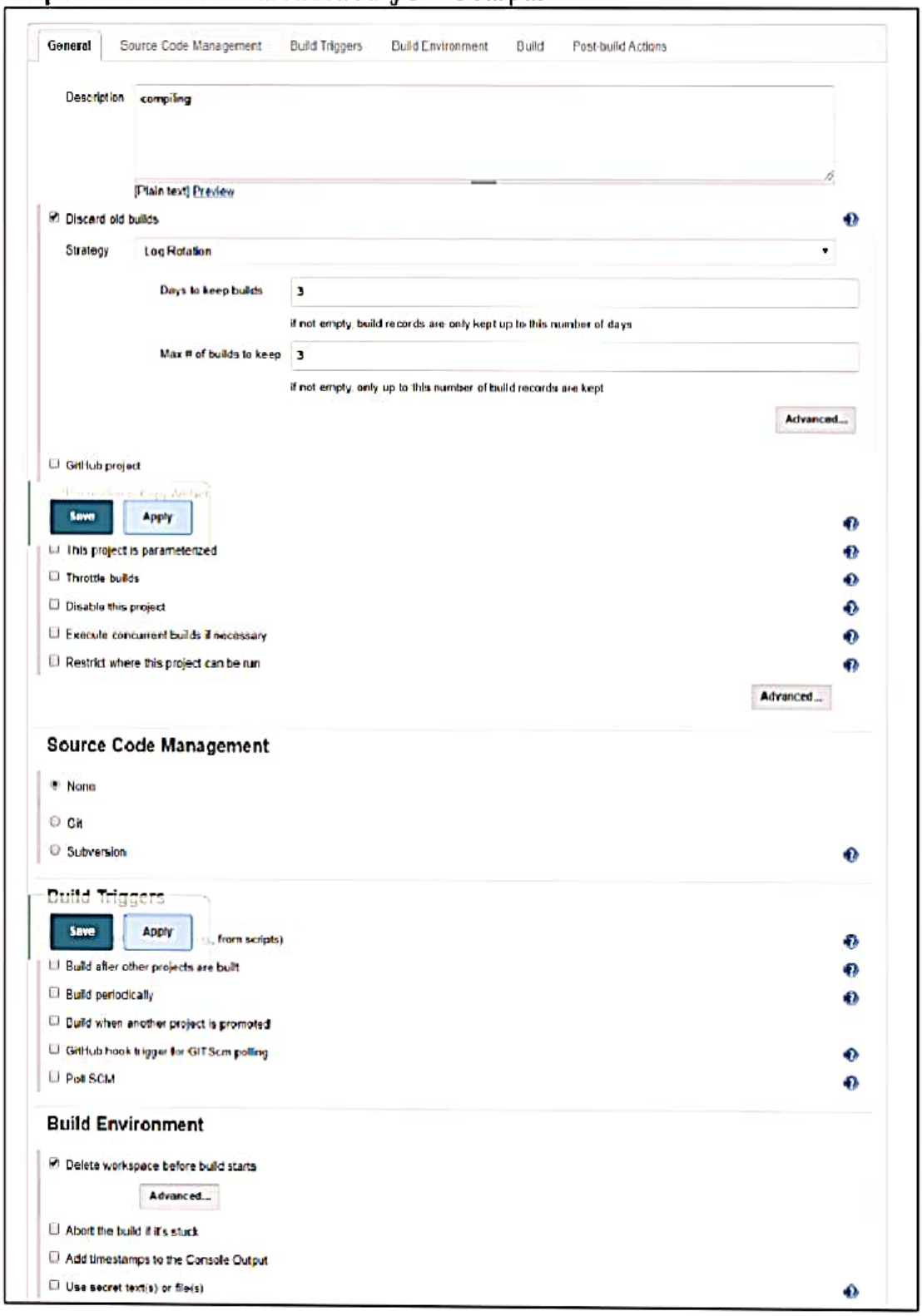


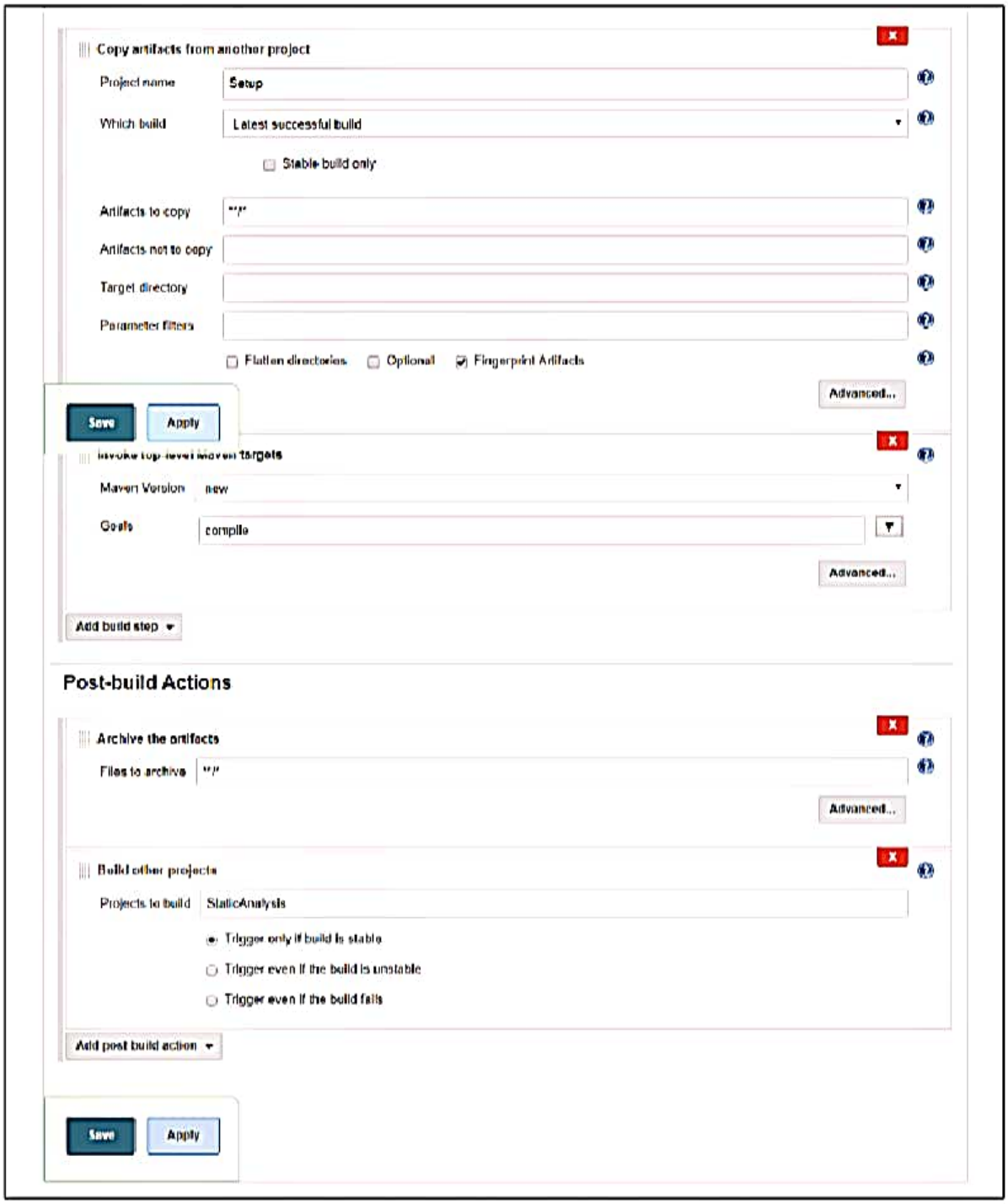

**Step 1:** Create below mentioned job - Setup


General
Source Code Management
Build Triggers
Build Environment
Build
Post-build Actions
Description
code check out
Discard old builds
Strategy
Log Rotation
Days to keep builds
Max # of builds to keep
Advanced...
GitHub project
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

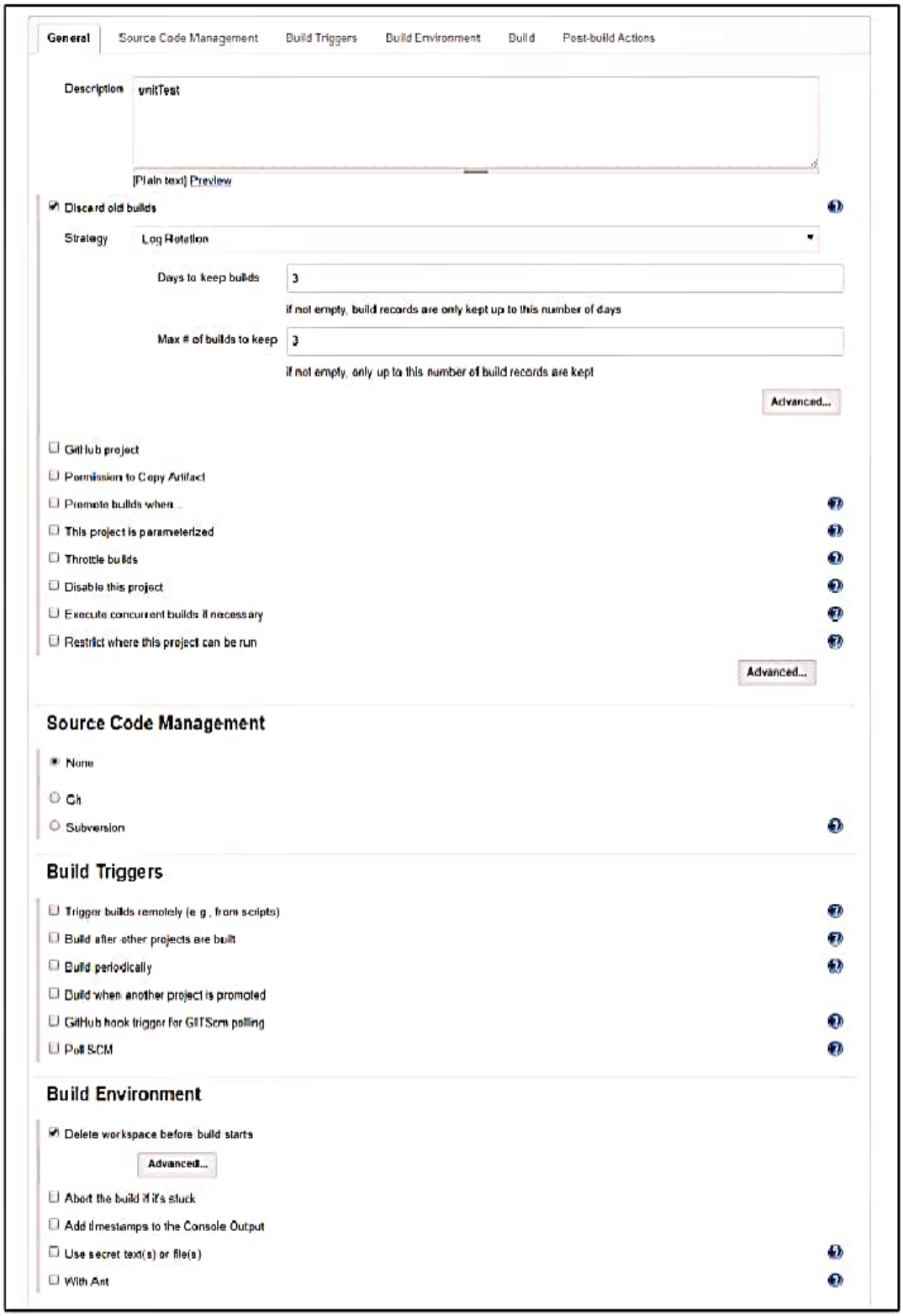

Build Triggers
Trigger builds remotely (e.g., from scripts)
Build after other projects are built
Build periodically
Build when another project is promoted
GitHub hook trigger for GITScm polling
Poll SCM
Schedule
H/2 * * * *
Ignore post-commit hooks
Build Environment
Delete workspace before build starts
Advanced...
Save
Apply
With Ant
Build
Invoke top-level Maven targets
Maven Version
maven
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

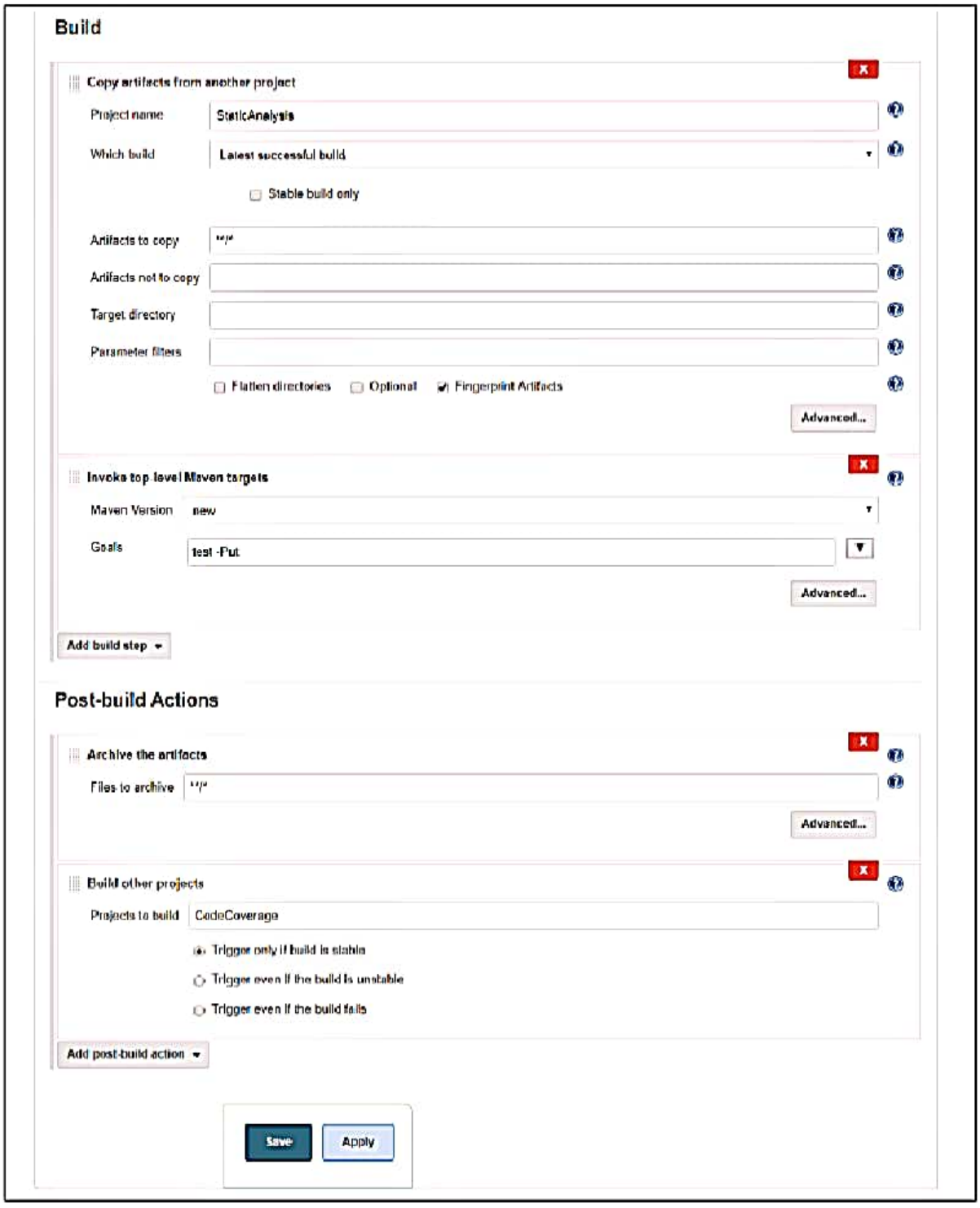
**Step 2:** Create below mentioned job – Compile



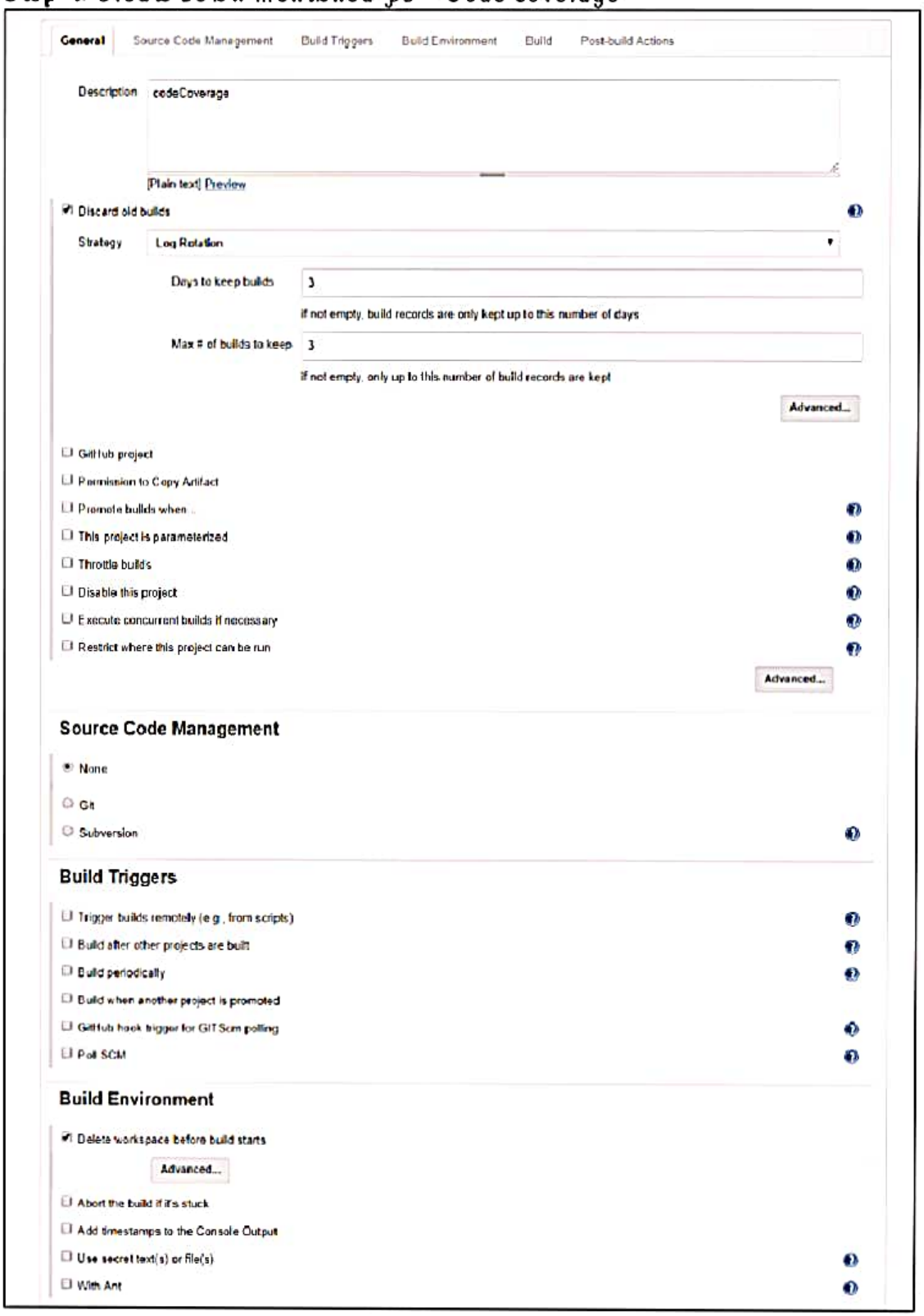

Copy artifacts from another project
Project name
Setup
Which build
Latest successful build
Stable build only
Artifacts to copy
**/*
Artifacts not to copy
Target directory
Parameter filters
Flatten directories
Optional
Fingerprint Artifacts
Advanced...
Save
Apply
Maven Version
new
Goals
compile
Advanced...
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

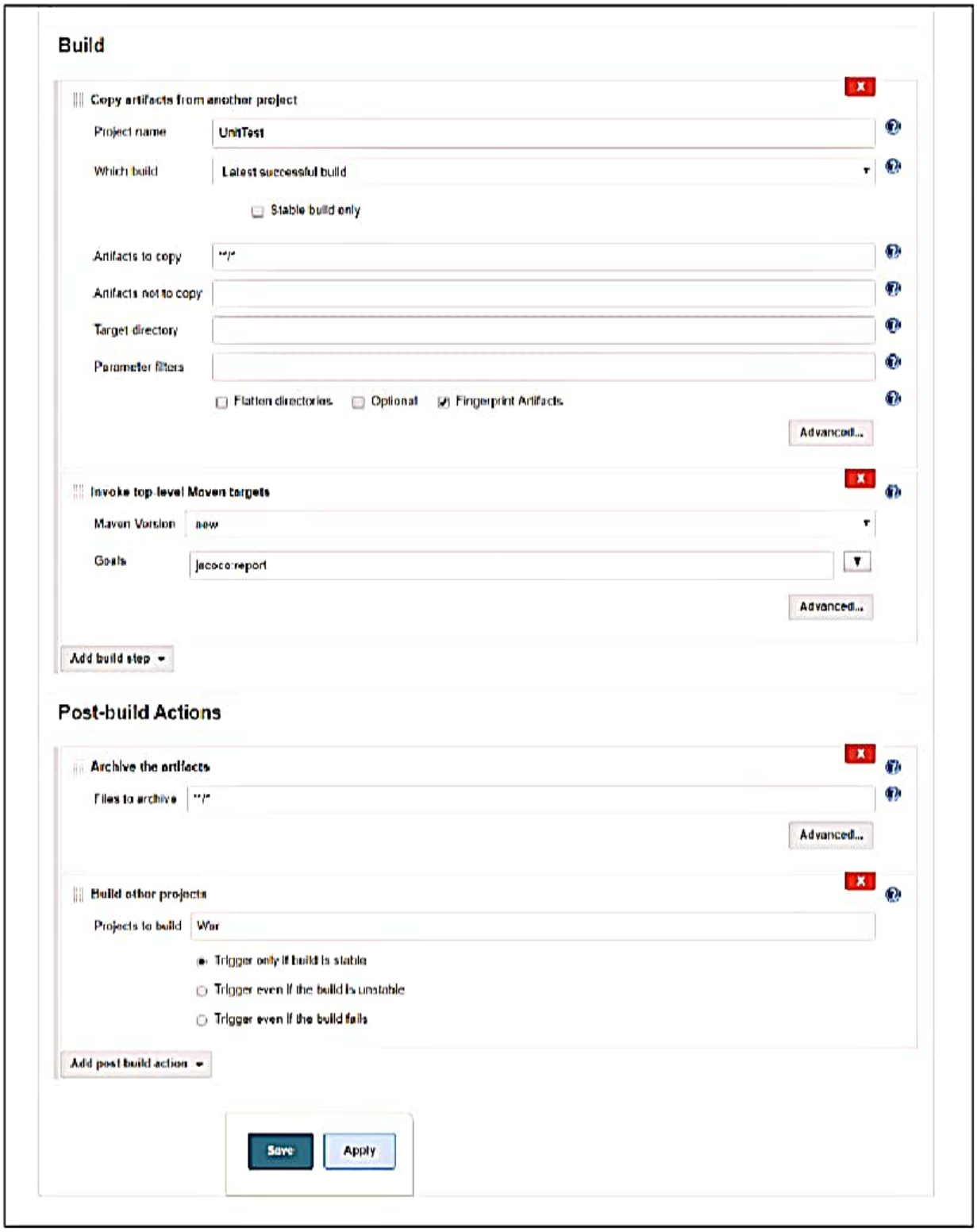
**Step 3:** Create below mentioned job – Unit test

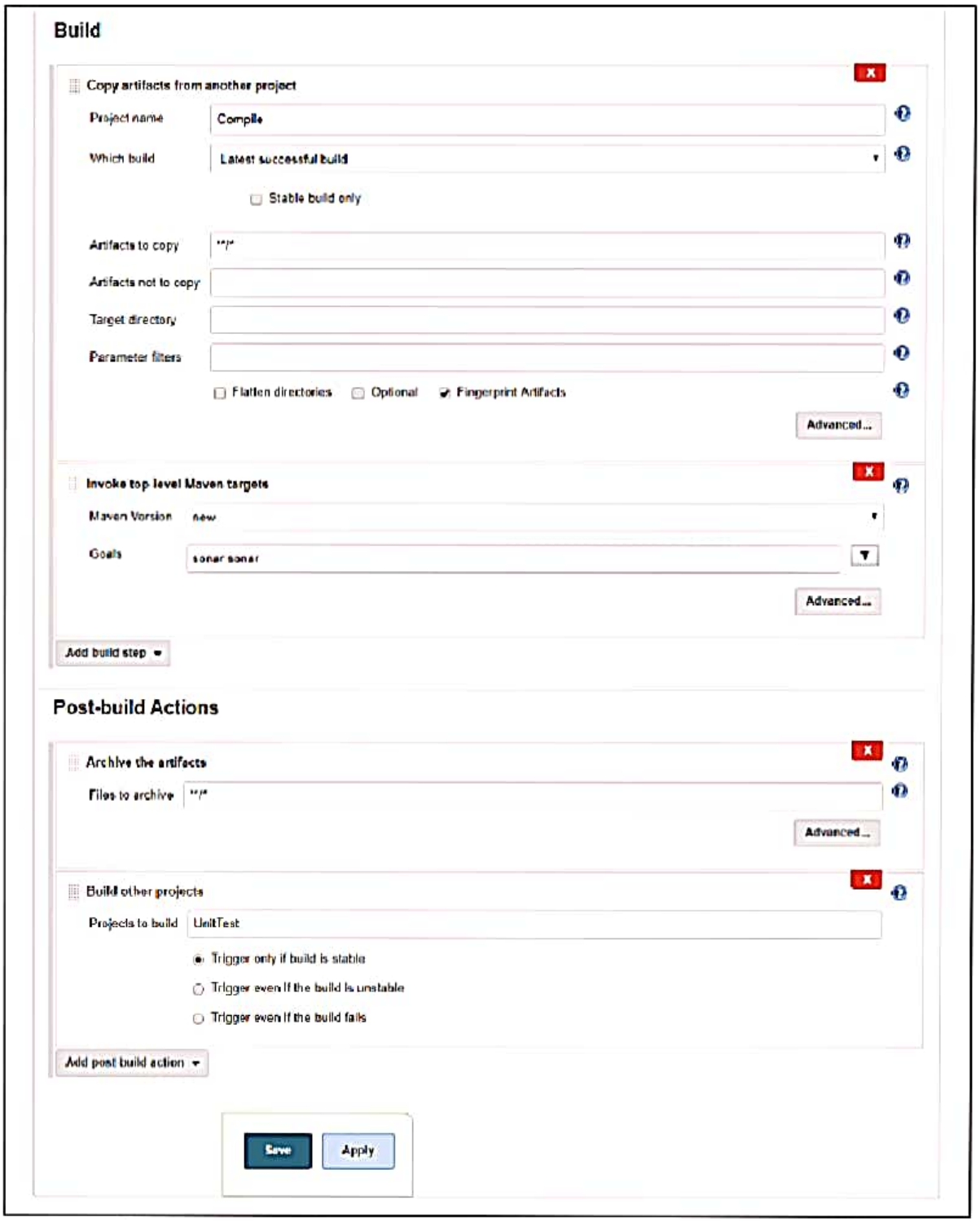



Build
Copy artifacts from another project
Project name
StaticAnalysis
Which build
Latest successful build
Stable build only
Artifacts to copy
**/*
Artifacts not to copy
Target directory
Parameter filters
Flatten directories
Optional
Fingerprint Artifacts
Advanced...
Invoke top-level Maven targets
Maven Version
new
Goals
test -Put
Advanced...
Add build step
Post-build Actions
Archive the artifacts
Files to archive
**/*
Advanced...
Build other projects
Projects to build
CodeCoverage
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

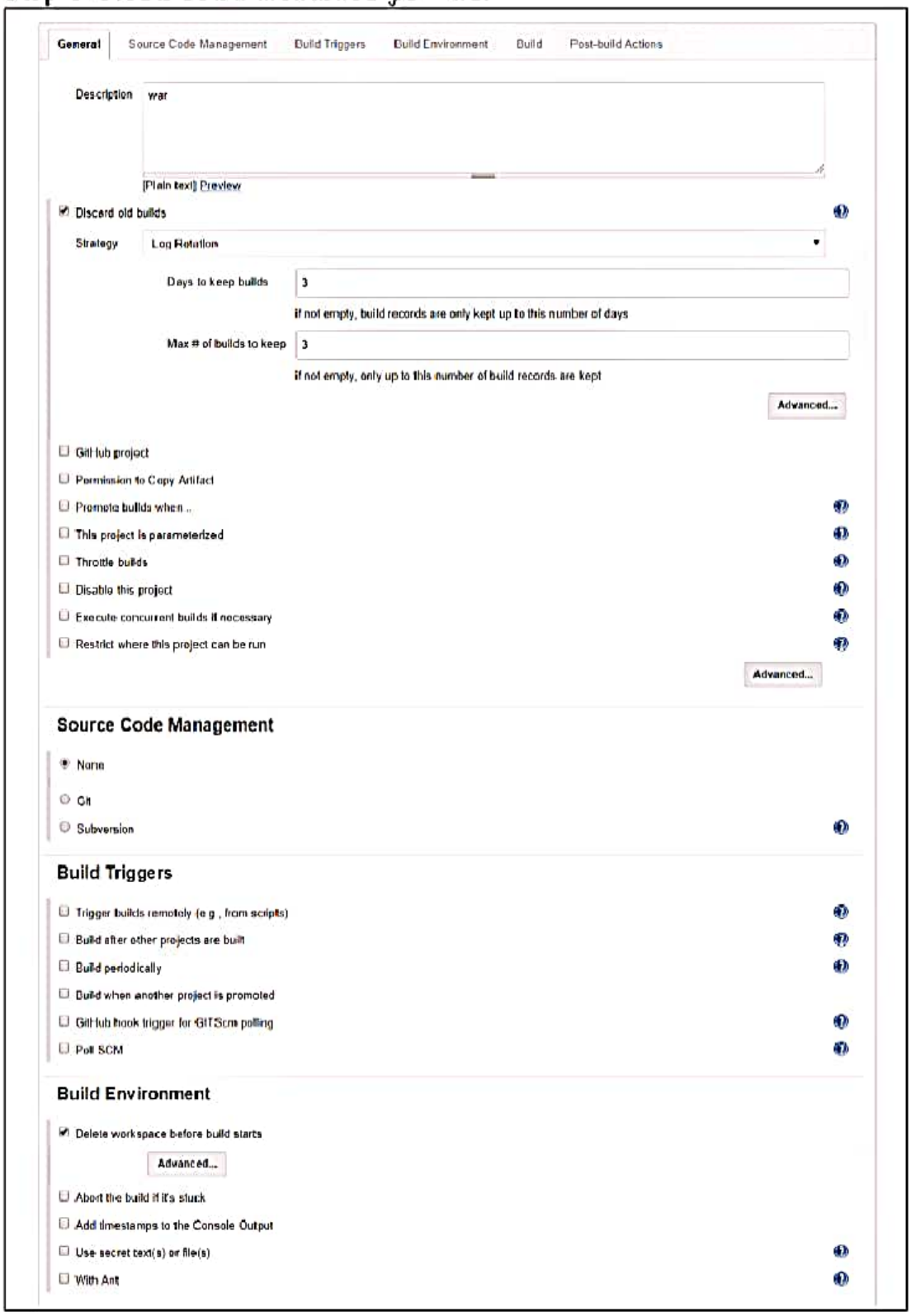
**Step 4:** Create below mentioned job – Code coverage

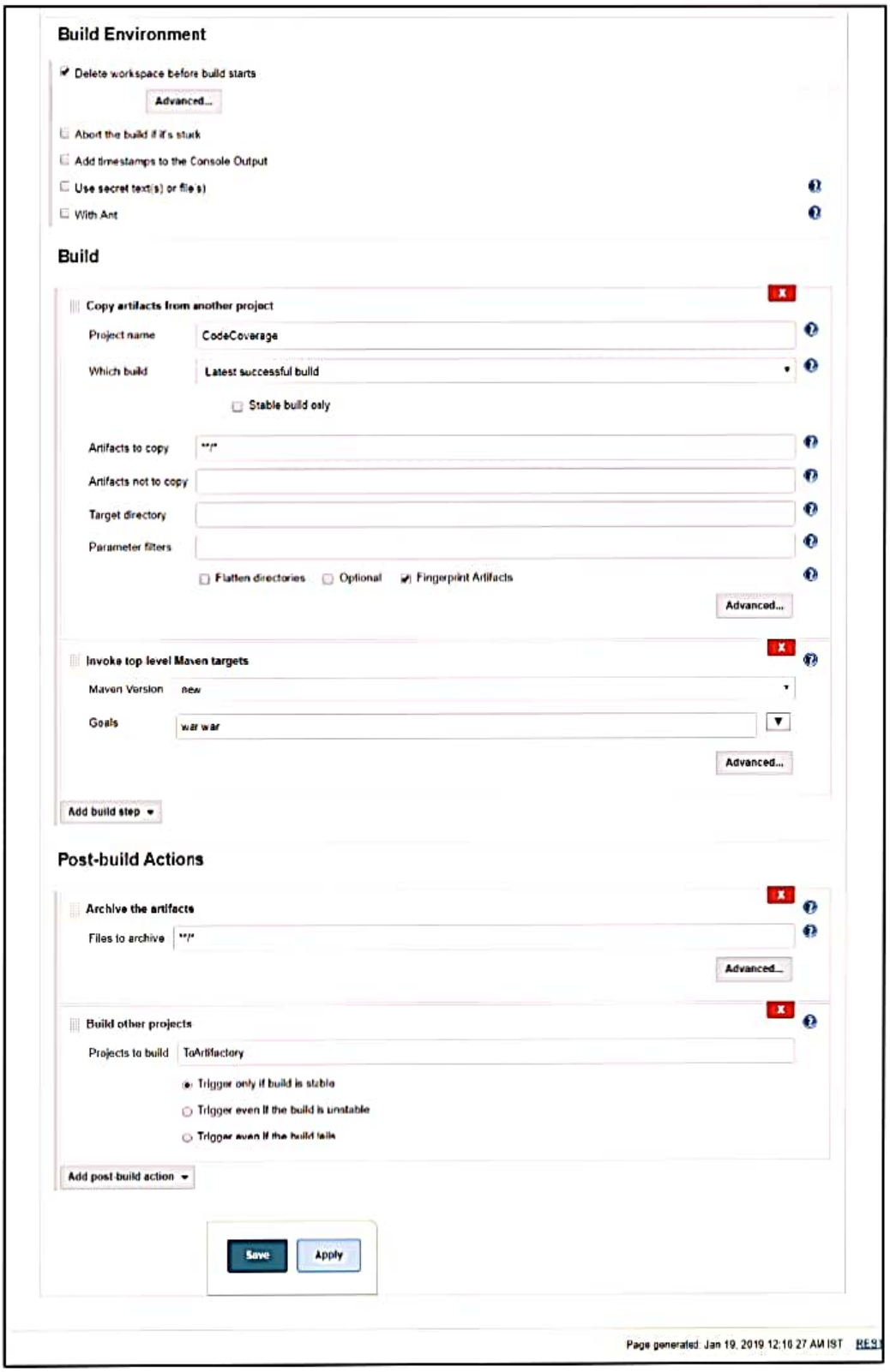



Build
Copy artifacts from another project
Project name
UnitTest
Which build
Latest successful build
Stable build only
Artifacts to copy
**/*
Artifacts not to copy
Target directory
Parameter filters
Flatten directories
Optional
Fingerprint Artifacts
Advanced...
Invoke top-level Maven targets
Maven Version
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

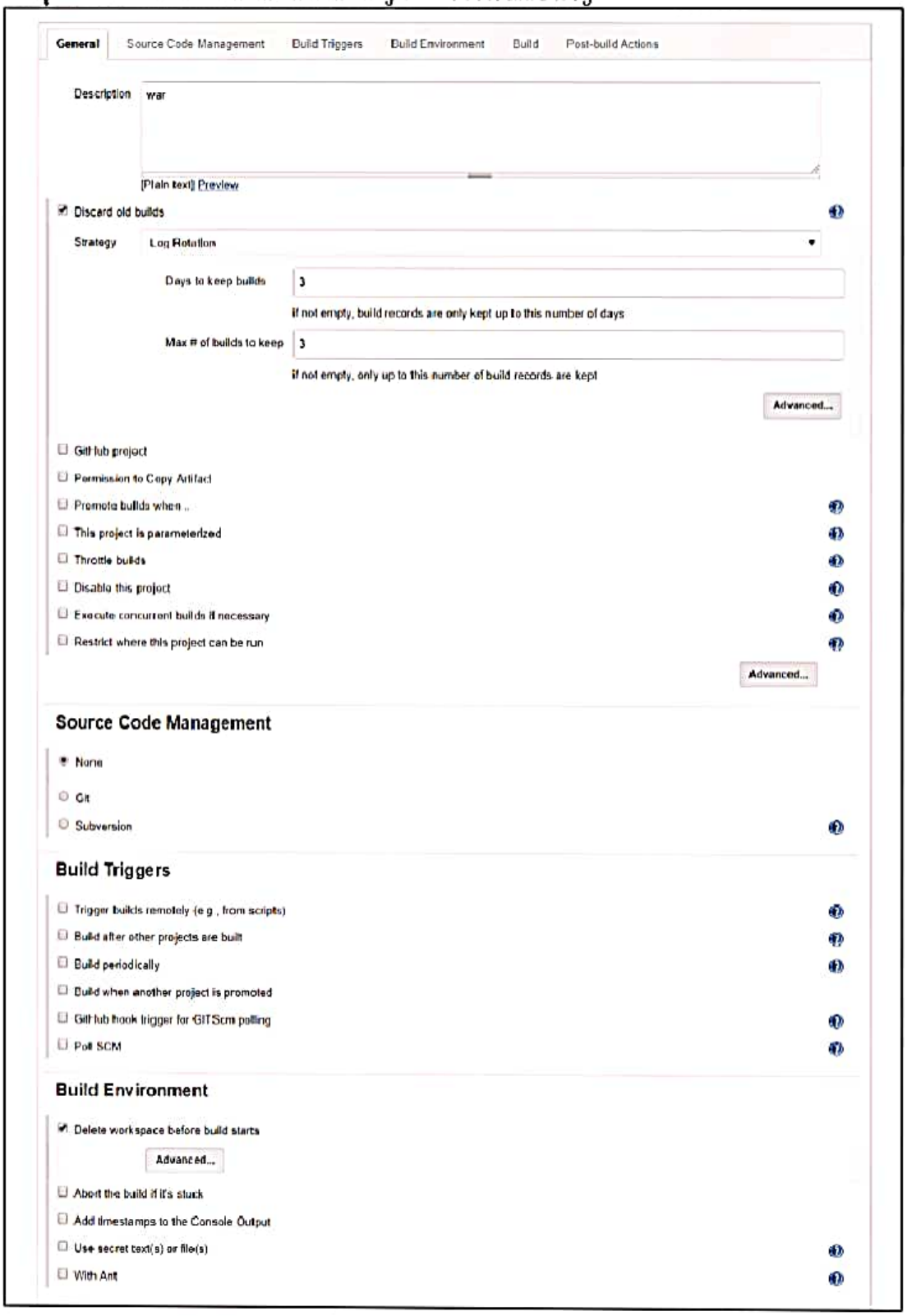

Build
Copy artifacts from another project
Project name
Compile
Which build
Latest successful build
Stable build only
Artifacts to copy
**/*
Artifacts not to copy
Target directory
Parameter filters
Flatten directories
Optional
Fingerprint Artifacts
Advanced...
Invoke top-level Maven targets
Maven Version
new
Goals
sonar:sonar
Advanced...
Add build step
Post-build Actions
Archive the artifacts
Files to archive
**/*
Advanced...
Build other projects
Projects to build
UnitTest
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

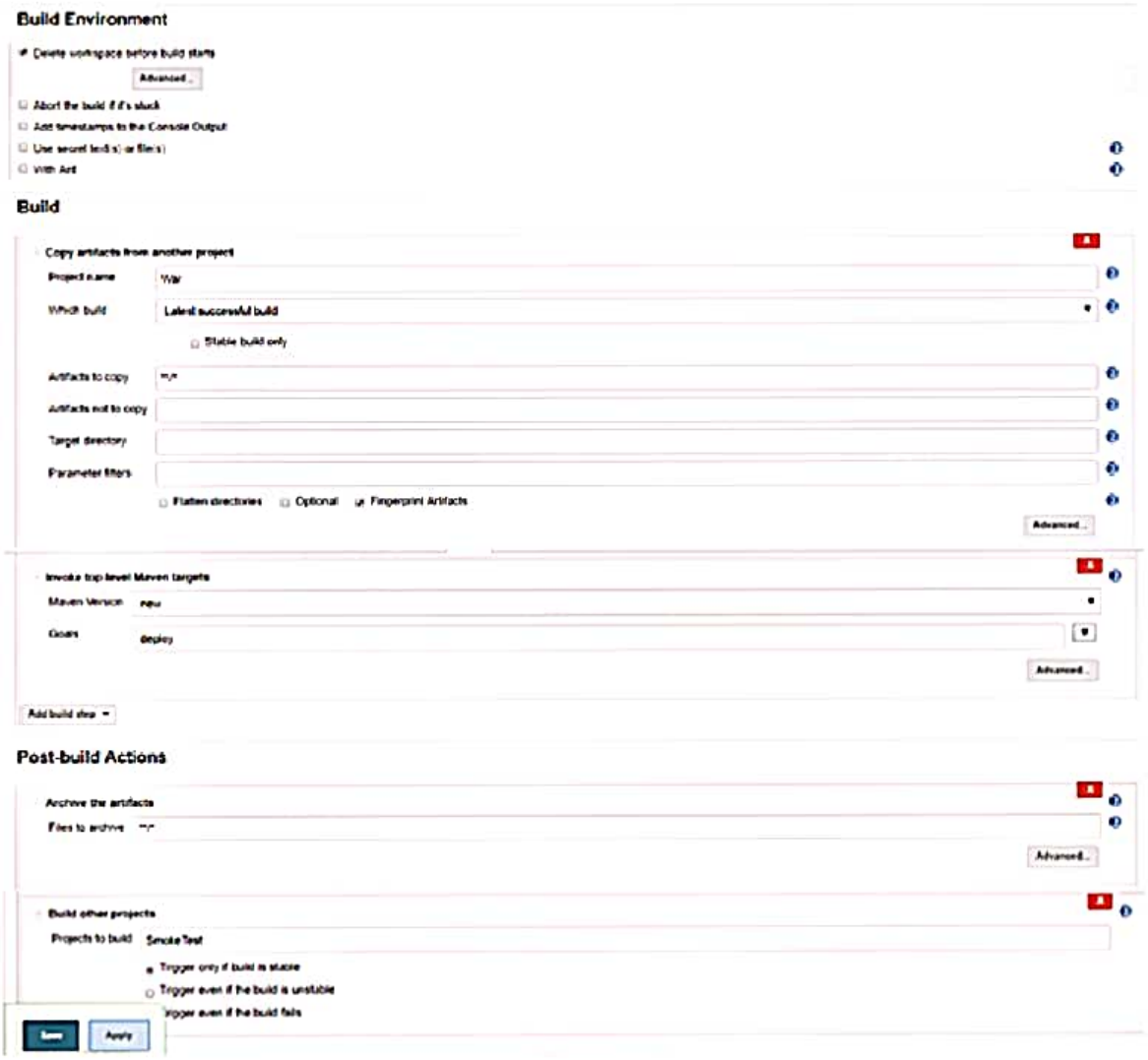
**Step 6:** Create below mentioned job – war



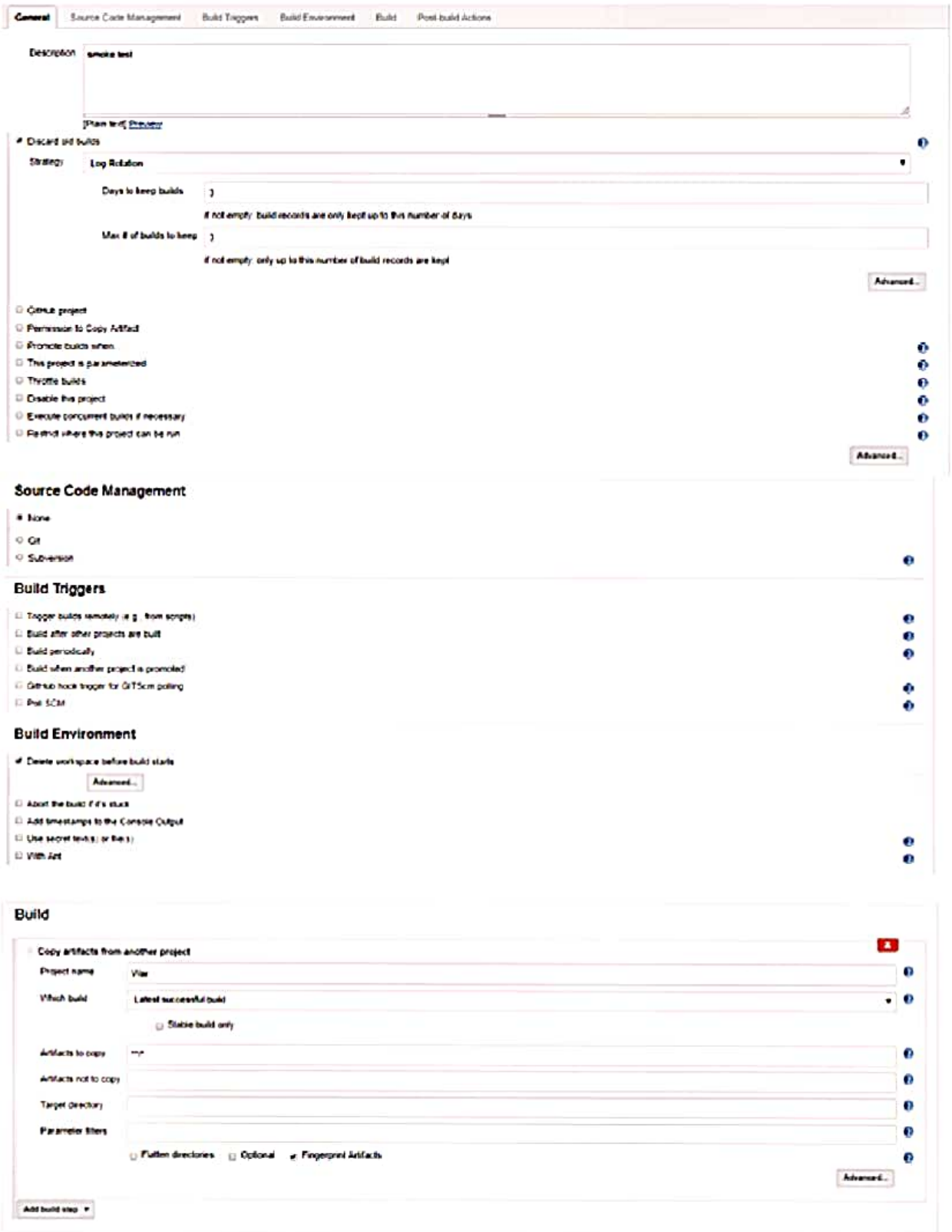

Build Environment
Delete workspace before build starts
Advanced...
Abort the build if it's stuck
Add timestamps to the Console Output
Use secret text(s) or file(s)
With Ant
Build
Copy artifacts from another project
Project name
CodeCoverage
Which build
Latest successful build
Stable build only
Artifacts 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
Post-build Actions
Archive the artifacts
Files to archive
**/*
Advanced...
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
Page generated: Jan 19, 2019 12:10:27 AM IST

**Step 7:** Create below mentioned job – ToArtifactory






**Step 8:** Create below mentioned job – SmokeTest


General
Source Code Management
Build Triggers
Build Environment
Build
Post-build Actions
Description
smoke test
Discard old builds
Strategy
Log Rotation
Days to keep builds
Max # of builds to keep
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
Source Code Management
None
Git
Subversion
Build Triggers
Build after other projects are built
Build periodically
Build when another project is promoted
Poll SCM
Build Environment
Delete workspace before build starts
Abort the build if it's stuck
Add timestamps to the Console Output
With Ant
Build
Copy artifacts from another project
Project name
Which build
Latest successful build
Stable build only
Artifacts to copy
Artifacts not to copy
Target directory
Parameter filters
Flatten directories
Optional
Fingerprint Artifacts
Add build step

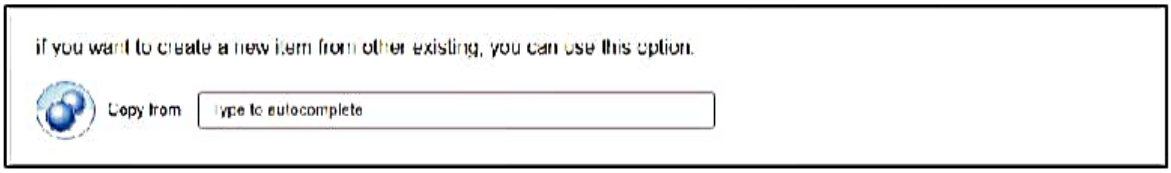

**Summary:** You learned main line CI pipeline creation.

## 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:




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.

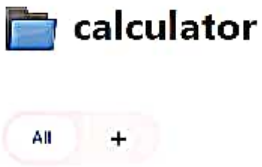

**Step 4**: Click on **Move** and observe the newly moved jobs in that location.
**Estimated Time:** 20 mins

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

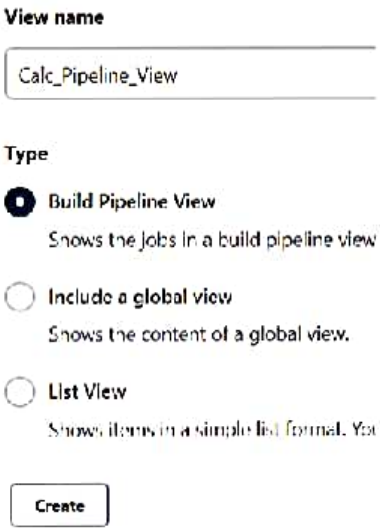
## Exercise 11: Creating pipeline view in Jenkins

**Objective:** Understand creation of pipeline view in Jenkins

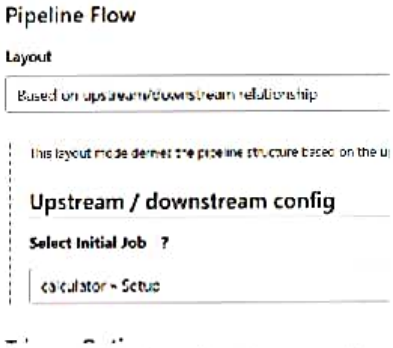
**Step 1**: Click on the '+' symbol under the Jenkins project folder as shown in the screenshot given below




**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.




**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.




**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

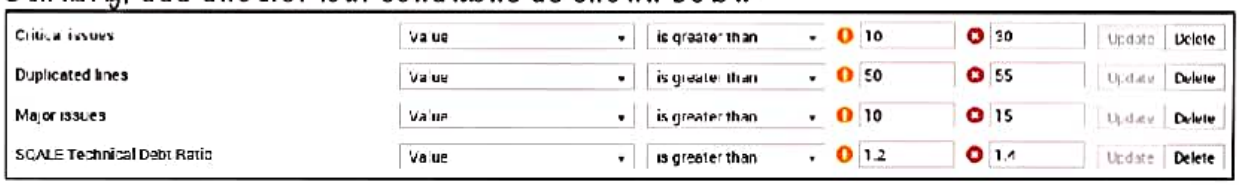
**Objective:** Configure gating conditions in Jenkins.

**Apply gates in the tool**

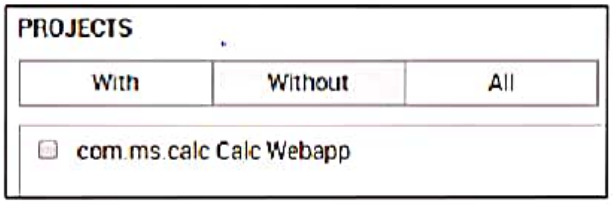
**Gating condition in Static Analysis job:**
**Step 1:** Start "SonarQube" server if not started earlier.

**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.

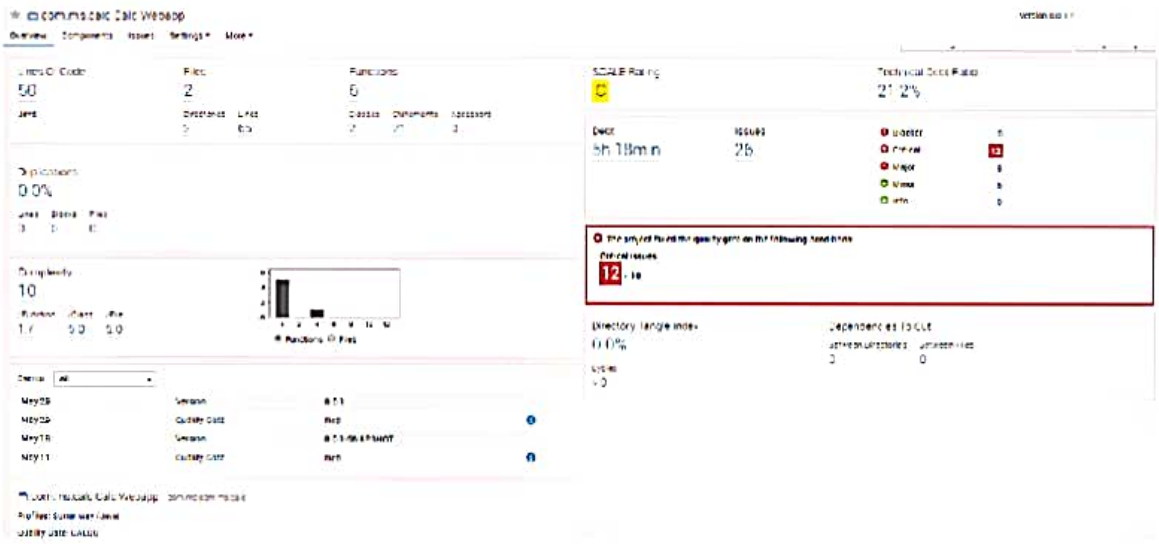

Similarly, add another four conditions as shown below




**Step 3:** To link the "CALQG" with project, use below shown option available in the same page (check the projectname in withoutsection, then that gate applied)



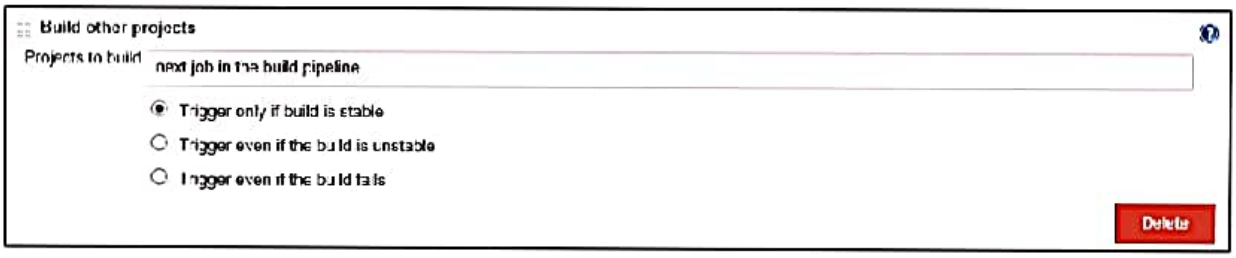

Once static analysis done (by running Maven Target) using SonarQube, you can observe the current analysis results on SonarQube dashboard as shown below.



**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.

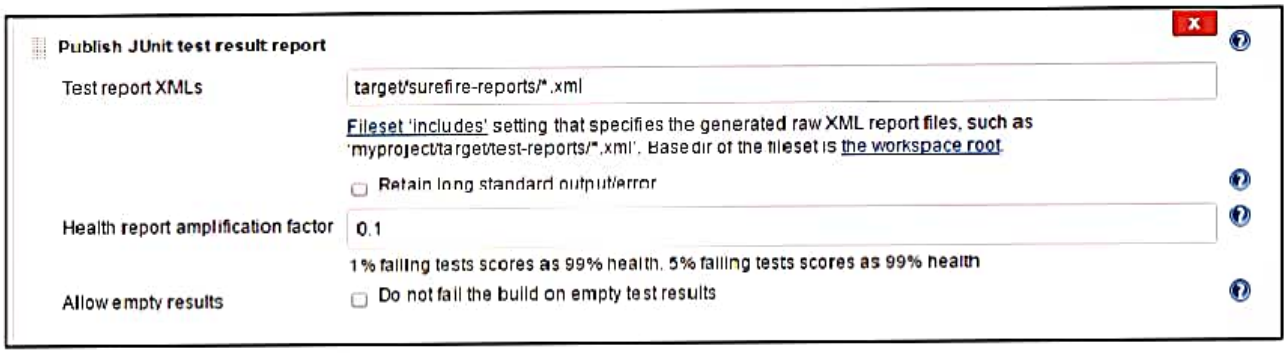



Note: You can configure default quality gates also in SonarQube.

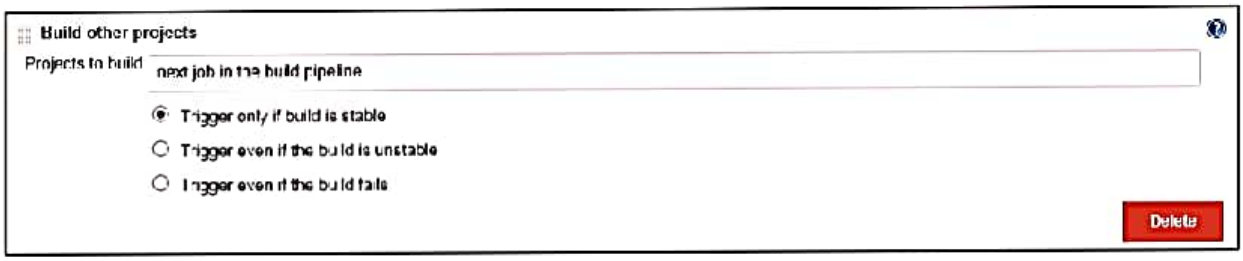
## Gating in Jenkins

**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);
}
```

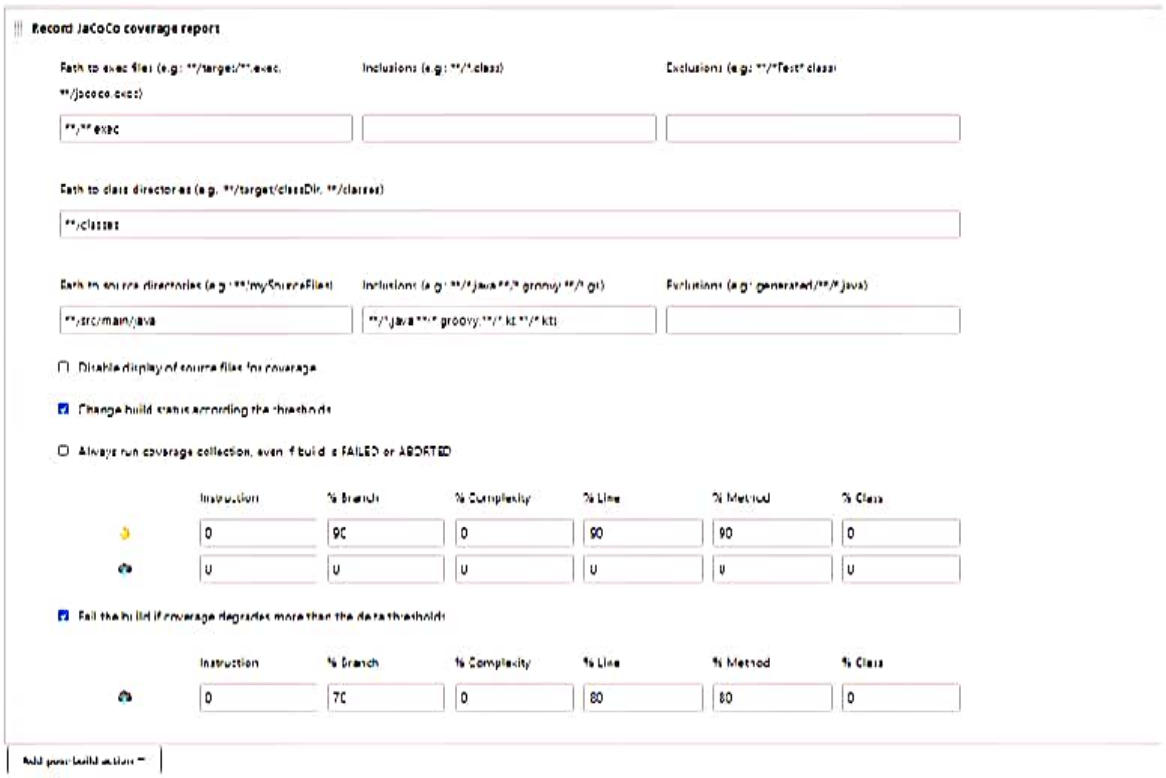
**Step 5:** Commit the code and check the Jenkins pipeline for the failure.

**Step 6:** You can also fail few more testcases 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.




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


**Here is a summary of the activities to be done**

a) 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

b) Trigger a build in Jenkins and observe the gating condition in SonarQube working.

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 

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

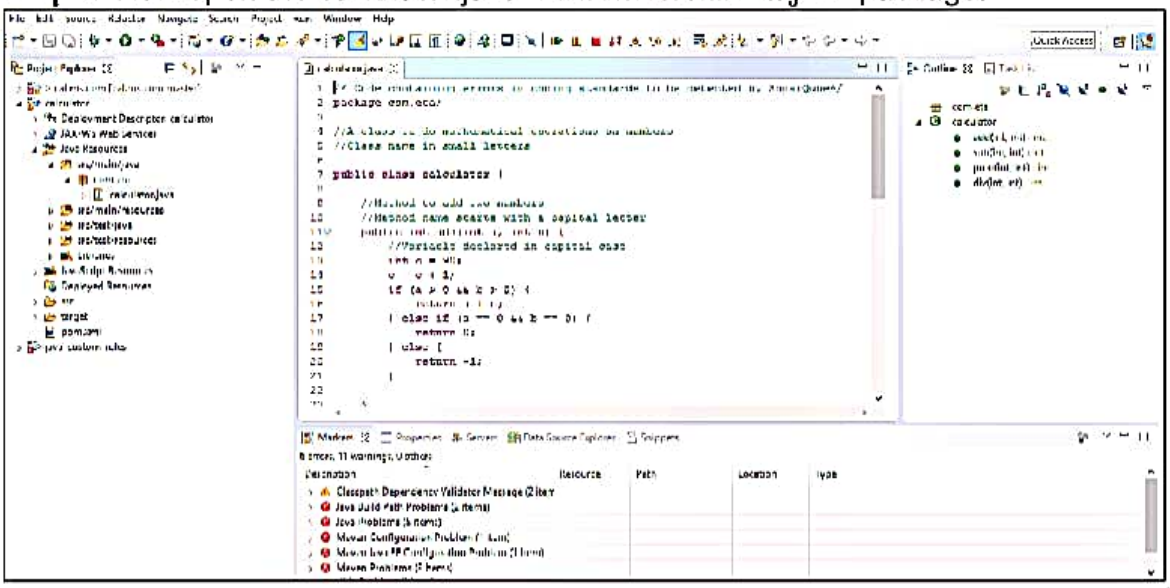
**Estimated time:** 30 mins

**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.



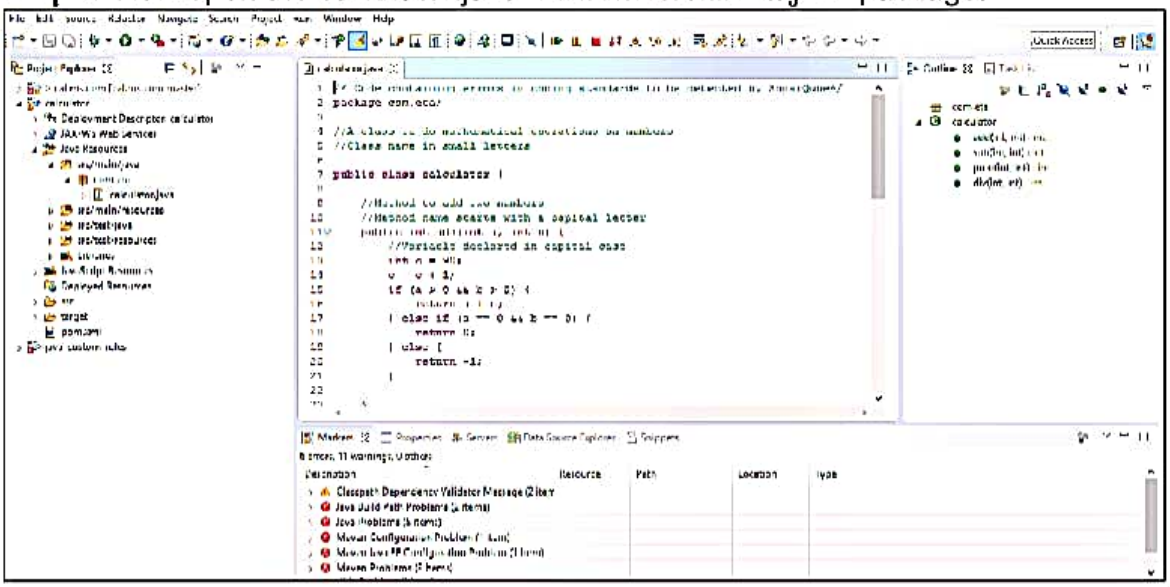
**Working with SonarLint:**

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

## 2. Static program analysis using SonarLint

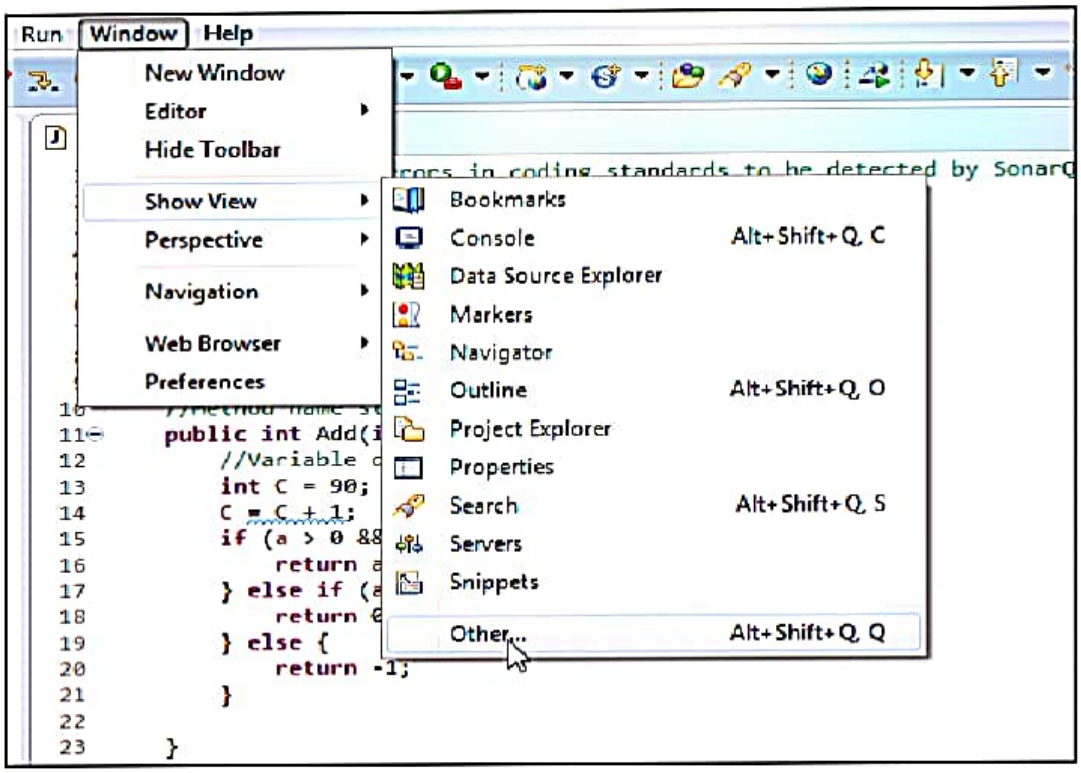
**Objective:** Perform static program analysis during coding using SonarLint plug-in to Eclipse IDE.
SonarLint is a plug-in to an IDE that provides on the fly feedback to developers on new bugs and quality issues injected into their code.

**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.

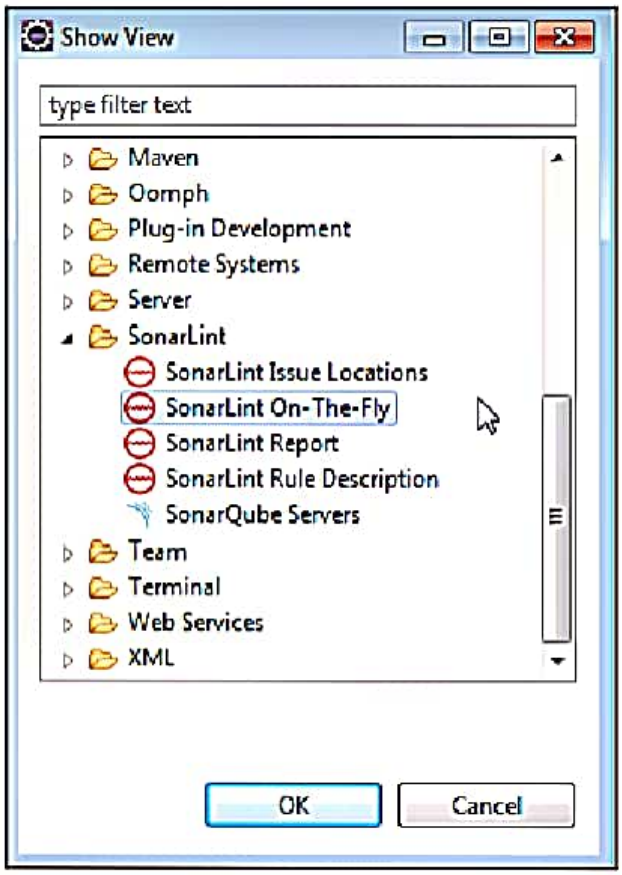


**Working with SonarLint:**

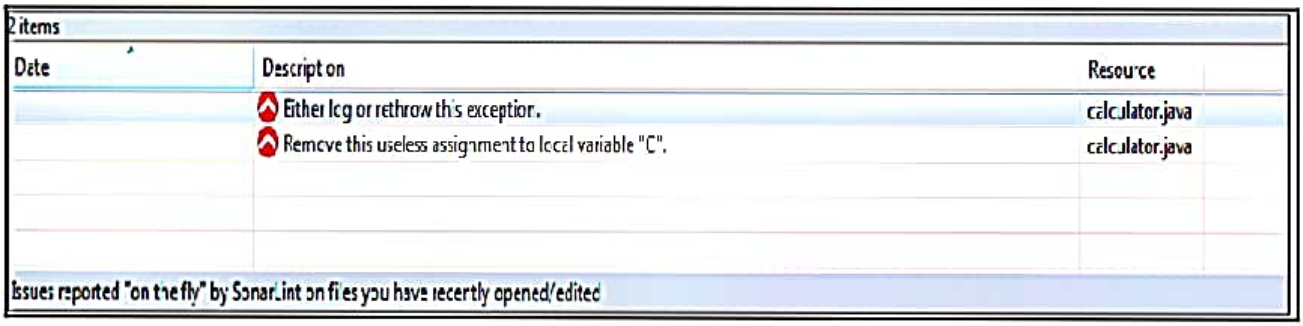
**Step 1:** Go to Windows>Show View>Other.



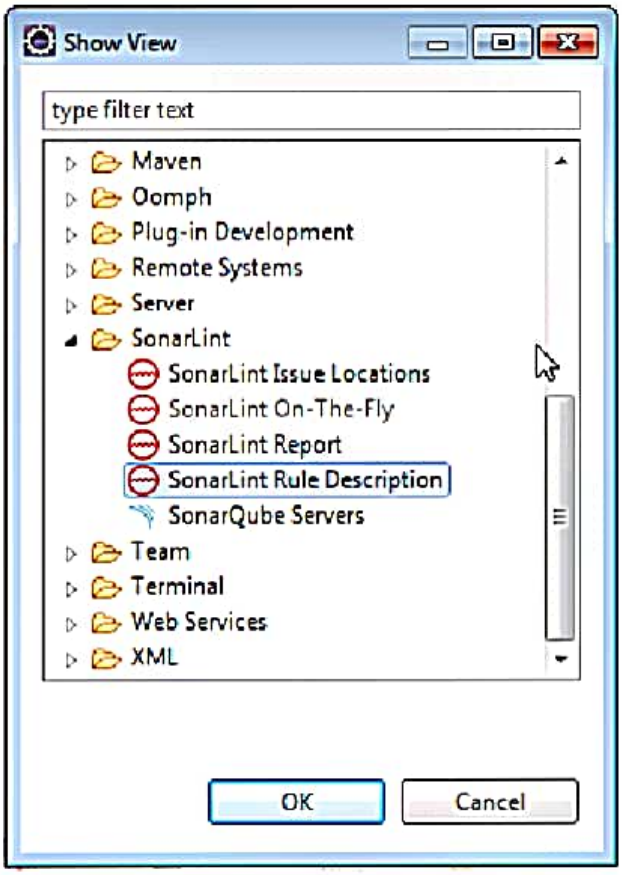

Here select SonarLint > SonarLint On-The-fly.



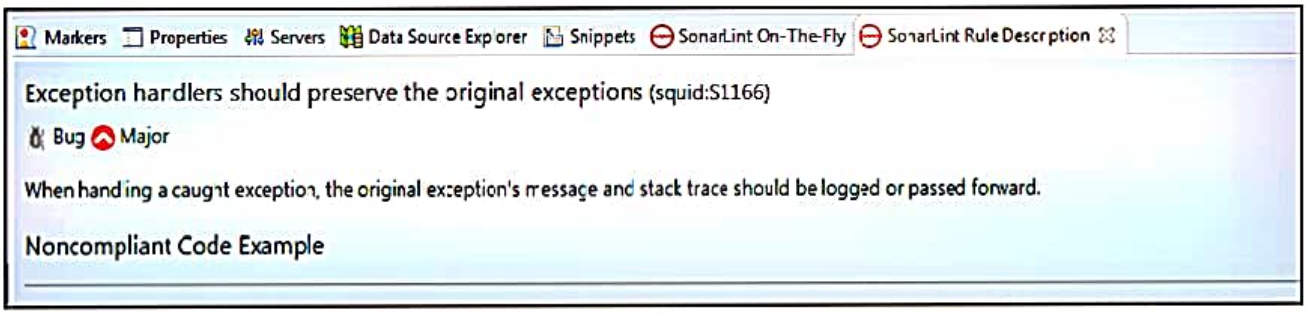

A new tab opens below which shows the issues found in the current java file selected.




**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

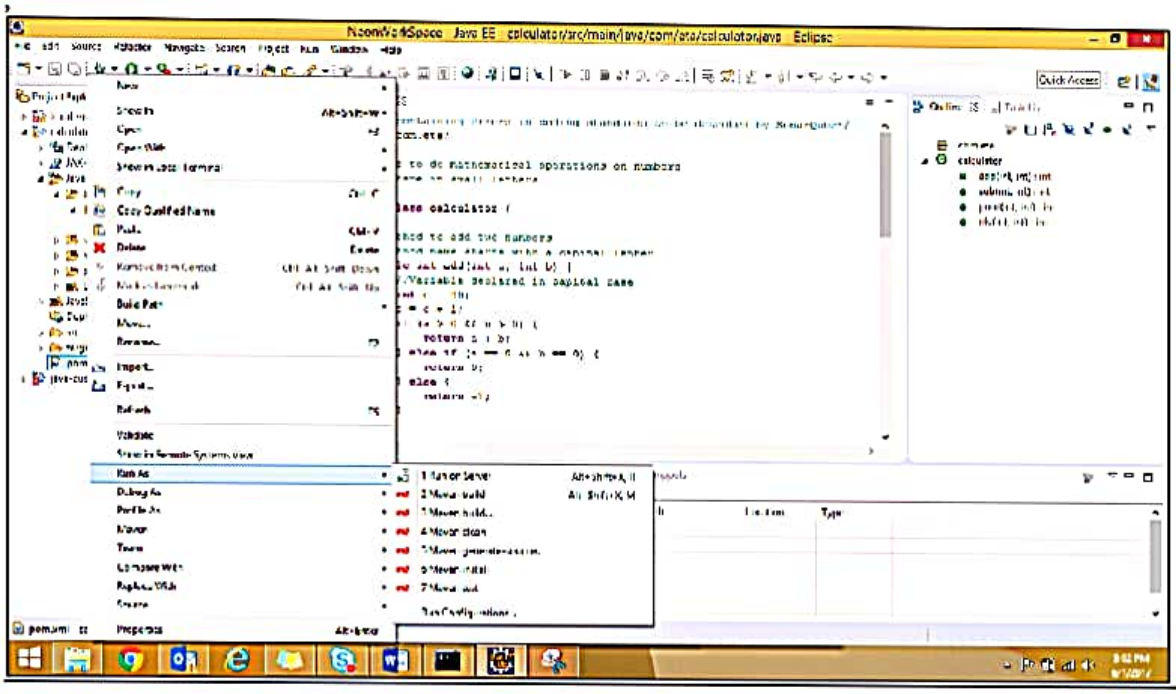
**Objective:** Customize the rules used by SonarLint through SonarQube web server interface.

**Requirements:** SonarQube 5.6 and above

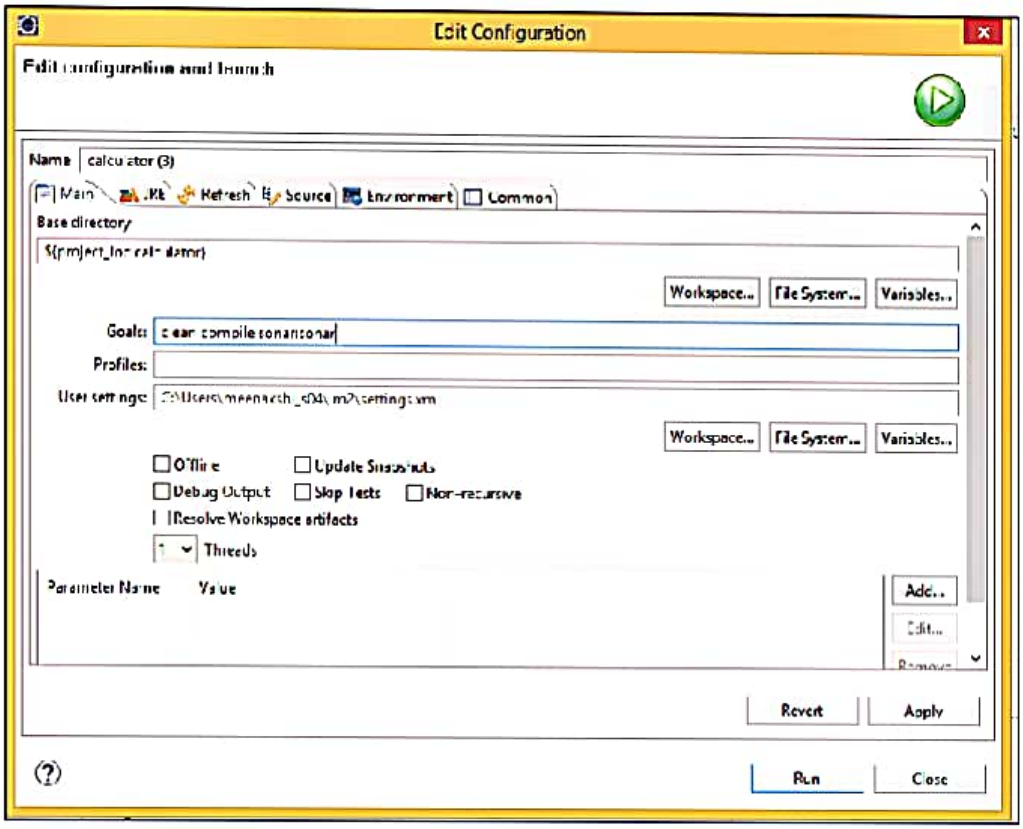
**Step 1:** Run the StartSonar.bat 

file as admin.

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

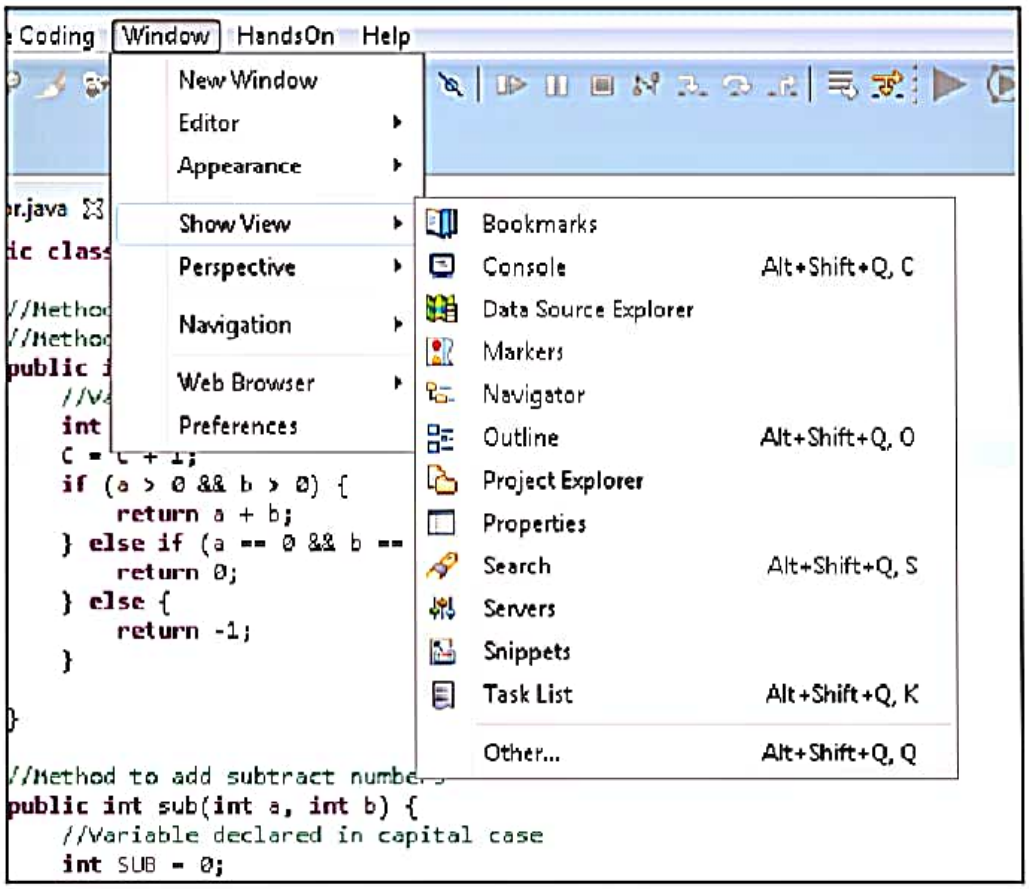


In the Goals tab, use clean, compile and sonar: sonar and execute the pom.xml file

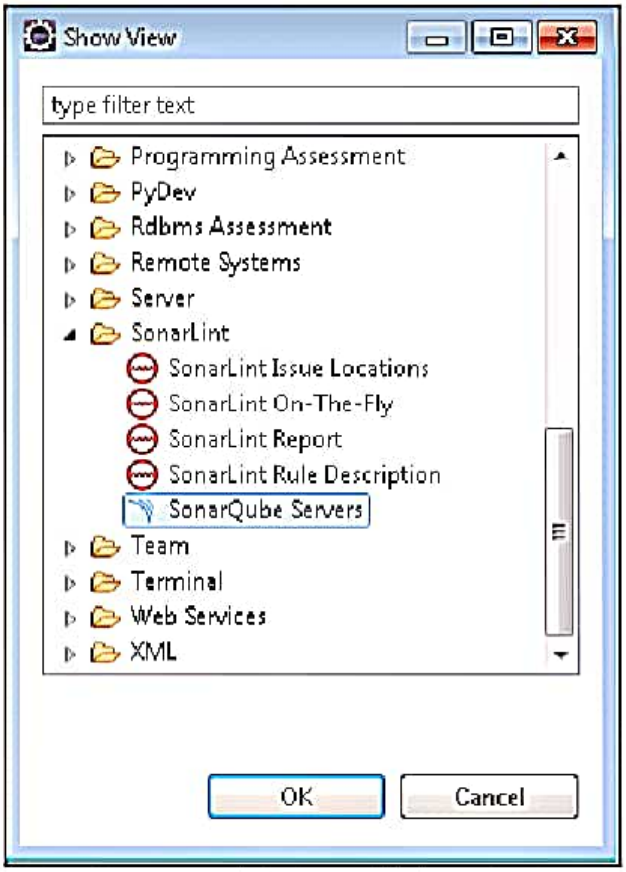



Once the build is successful, we need to bind the current java project with the SonarQube project.

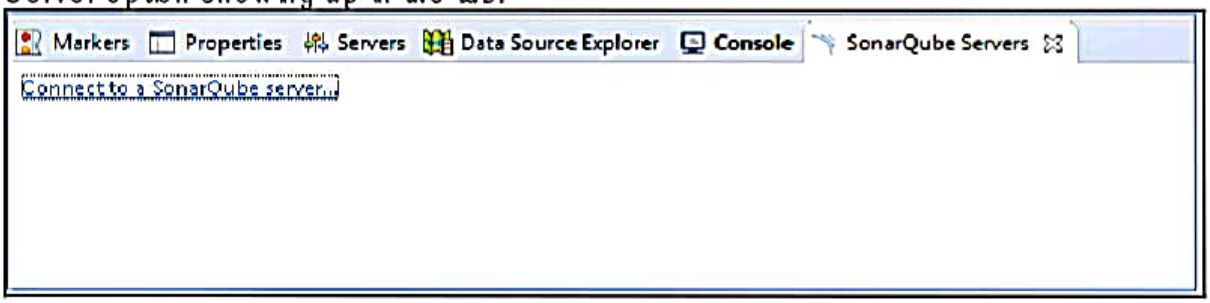
**Step 3:** Go to Window->Show View->Other.



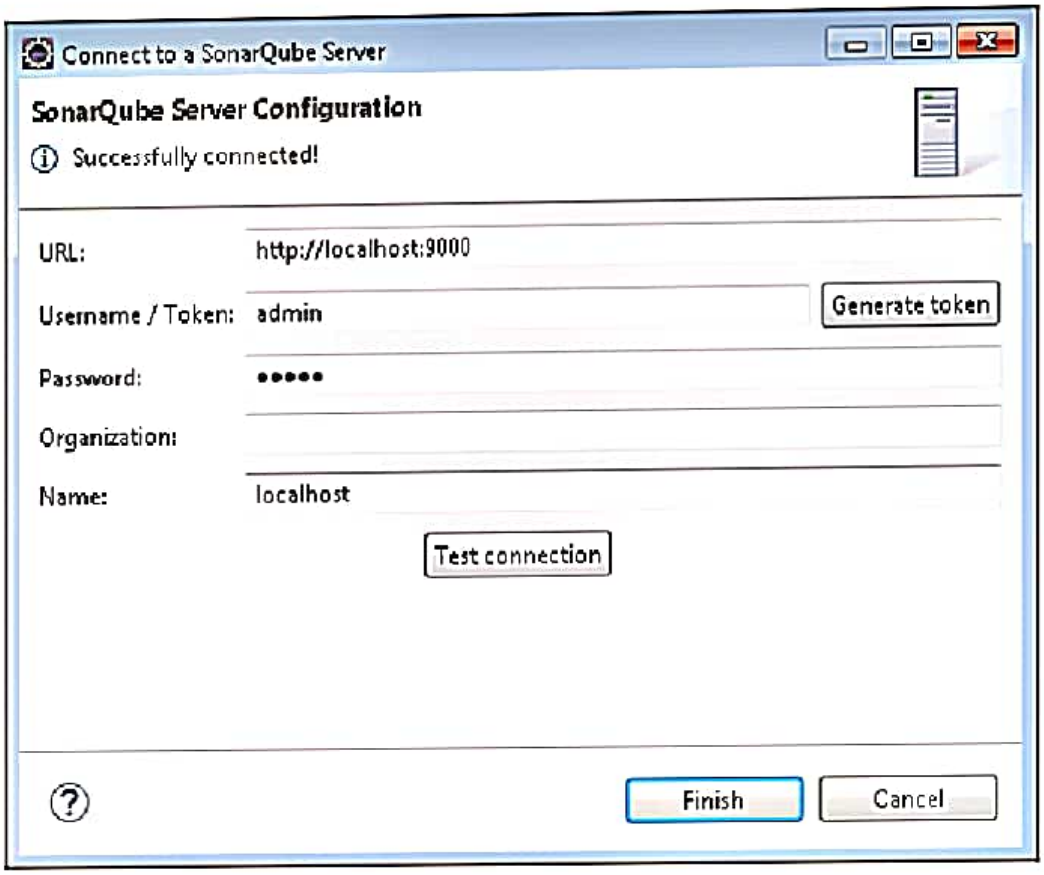

Select SonarLint-> SonarQube Server.



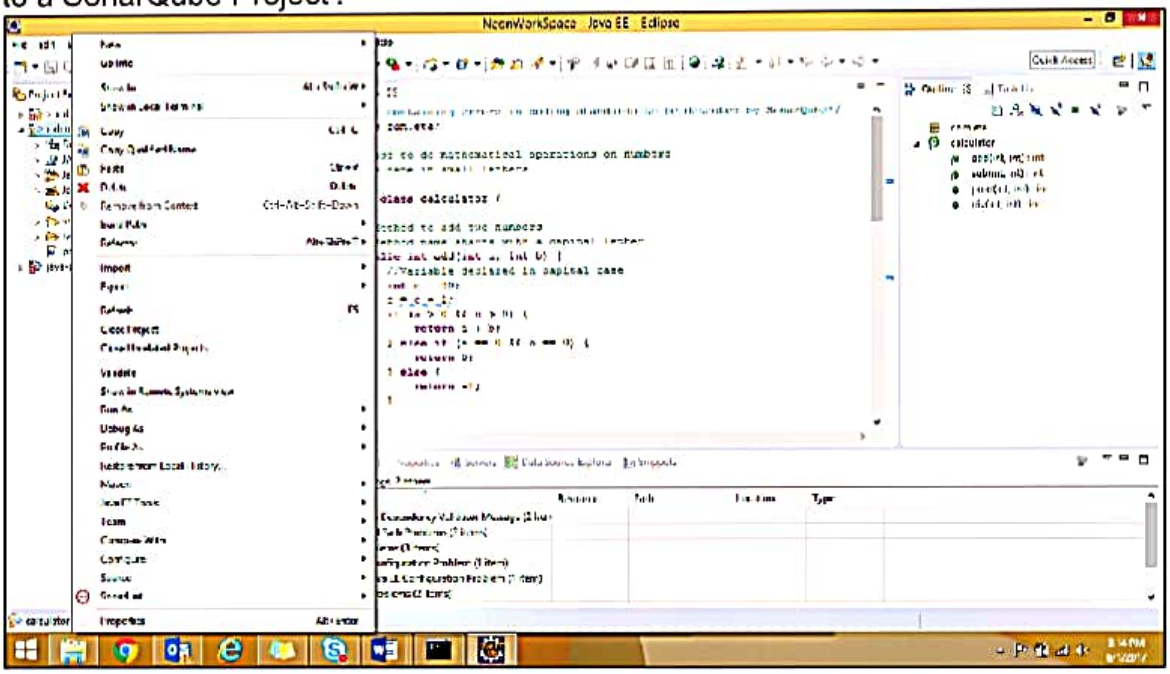

This opens a SonarQube Servers tab below. Click on the Connect to a SonarQube Server option showing up in the tab.



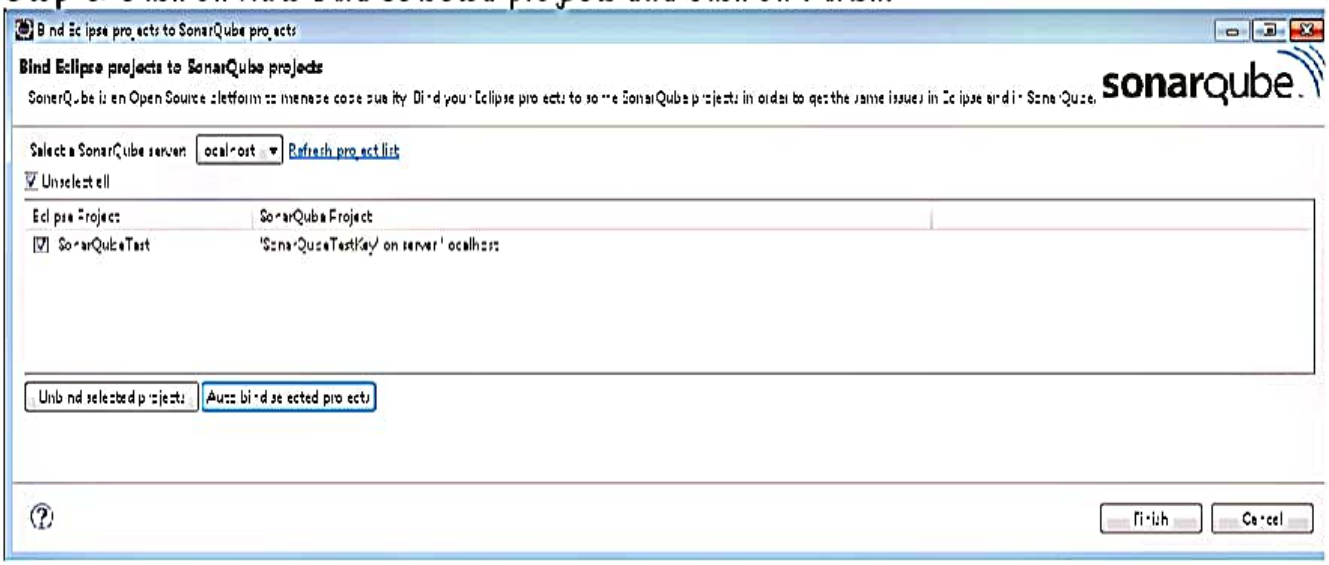

**Step 4:** Put the following details Username:admin Password:password as shown in the image given below and then click on TestConnection button.




**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.