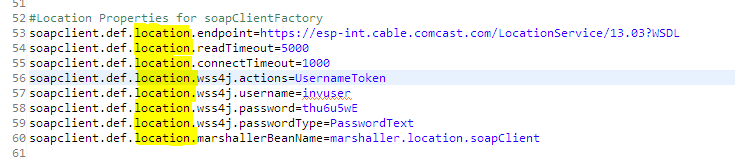
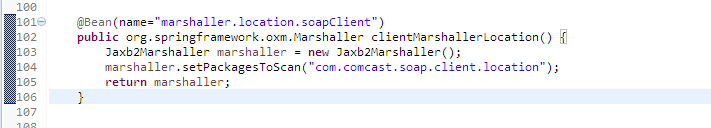
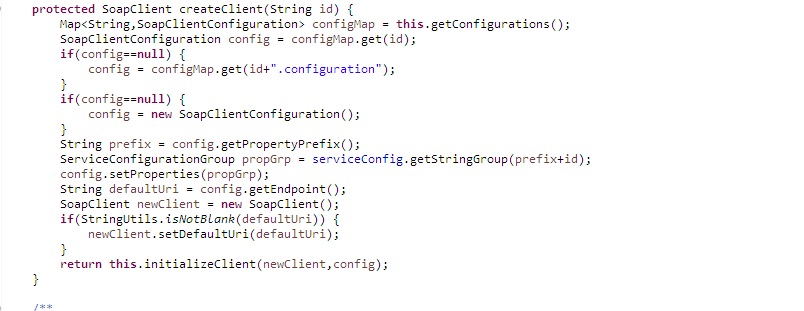
1. **Soap Client Factory:- Concept+ usage in project :**
2. **Adding parameters in application.properties for calling Location-Service :**



1. **Adding marshaller in PropertiesConfiguration.java for location Service Calling :**

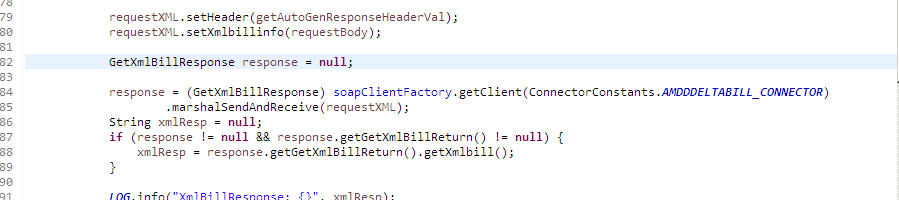
****

1. **SoapClientFactory detail for creating SoapClient for calling Location-Service :**

****

****

1. **Web Service calling using soapClient :**

****

1. **Junit Best practices(Mockito + Power Mock)**
2. Write 5 to 6 asserts at least.
3. Don’t handle exception in try catch block rather than use @expected when you are covering negative or catch block.
4. Follow camel case conversion like use test case names like testGetMarket rather than testgetMarket.
5. Never user Mocktio.any until and unless you are not able to cover.
6. Try to create seperate method for any object you are creating test case.
7. Always try to user real values in test cases like MarketId have value 451l.
8. Avoid unused variable declaration.
9. **Map-Struct- basics and application in our project**

**Gradle Dependency**

Add below line of code to your build.gradle file to add gradle dependency.

**buildscript** {

**repositories** {

mavenLocal()

maven {

url 'http://artifactory.io.comcast.net/plugins-release'

credentials {

username = "${artifactory\_user}"

password = "${artifactory\_password}"

}

}

}

**dependencies**{

classpath "net.ltgt.gradle:gradle-apt-plugin:0.5"

}

}

…………..

**apply** plugin: 'net.ltgt.apt'

**sourceCompatibility** = 1.8

…………

//mapstruct dependancy

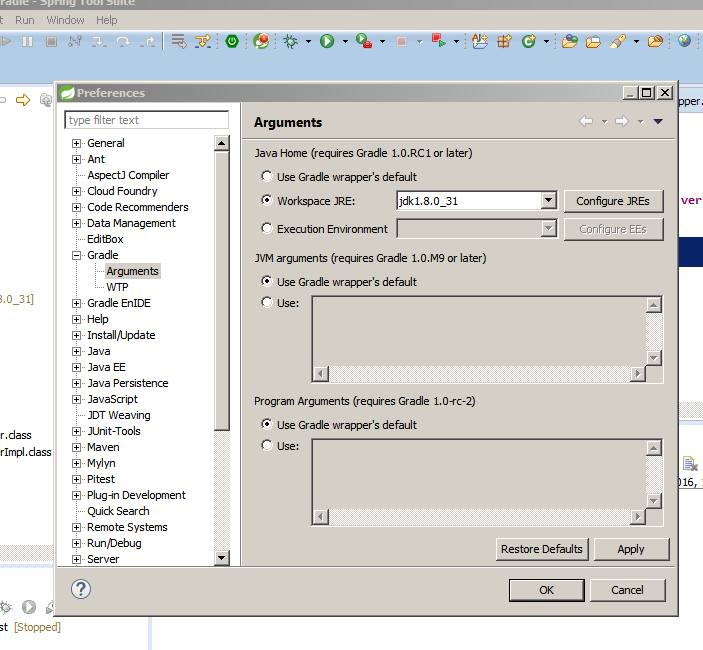
compile("org.mapstruct:mapstruct:1.0.0.Final")

apt 'org.mapstruct:mapstruct-processor:1.0.0.Final'

**Configuration with eclipse**

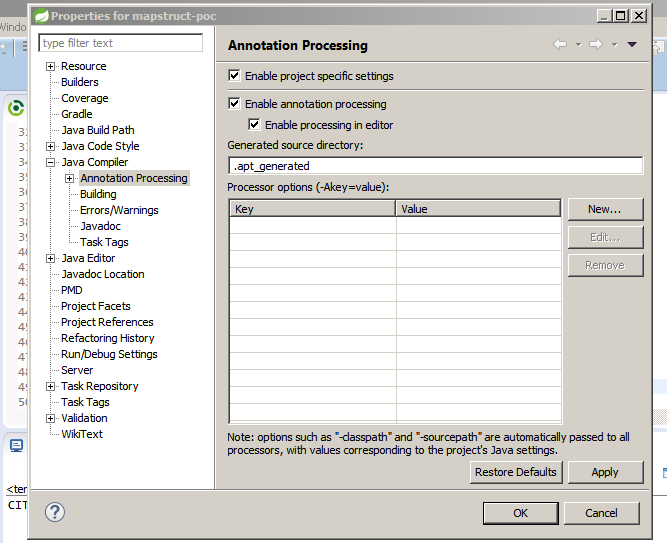
1. Click on window->preferences tab.
2. Click on Arguments inside Gradle

You will see below screen-



1. Select Workspace JRE and set there JDK instead of JRE. Then click Apply and OK.
2. Right click on project and then click on “Properties” option where u can find “Annotation processing “inside “Java Compiler” option.

You will get below screen

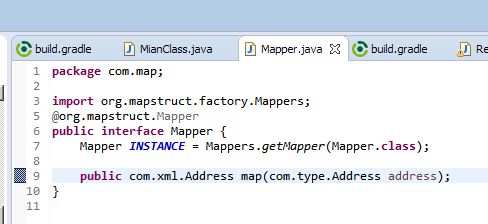


1. Select all checkbox and then Apply and OK.

Now all is done.

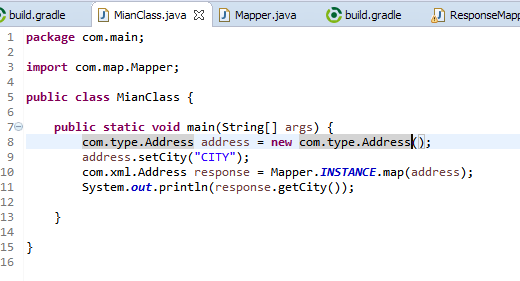
**A Sample code for mapstruct implementation**

In order to map two been you need to declare an interface. Please find sample code below-



Now you can see its implementation class inside “.apt\_generated” folder.

Please find sample code for mapping-



**Please find a sample “build.gradle” with mapstruct implementation.**

**buildscript** {

**repositories** {

mavenLocal()

maven {

url 'http://artifactory.io.comcast.net/plugins-release'

credentials {

username = "${artifactory\_user}"

password = "${artifactory\_password}"

}

}

}

**dependencies**{

classpath "net.ltgt.gradle:gradle-apt-plugin:0.5"

}

}

**apply** plugin: 'java'

**apply** plugin: 'eclipse'

**apply** plugin: 'net.ltgt.apt'

**sourceCompatibility** = 1.8

**version** = '1.0'

jar {

**manifest** {

attributes 'Implementation-Title': 'Gradle Quickstart', 'Implementation-Version': **version**

}

}

**repositories** {

mavenCentral()

}

**dependencies** {

compile **group**: 'commons-collections', **name**: 'commons-collections', **version**: '3.2'

testCompile **group**: 'junit', **name**: 'junit', **version**: '4.+'

//mapstruct dependancy

compile("org.mapstruct:mapstruct:1.0.0.Final")

apt 'org.mapstruct:mapstruct-processor:1.0.0.Final'

}

**test** {

systemProperties 'property': 'value'

}

uploadArchives {

**repositories** {

flatDir {

dirs 'repos'

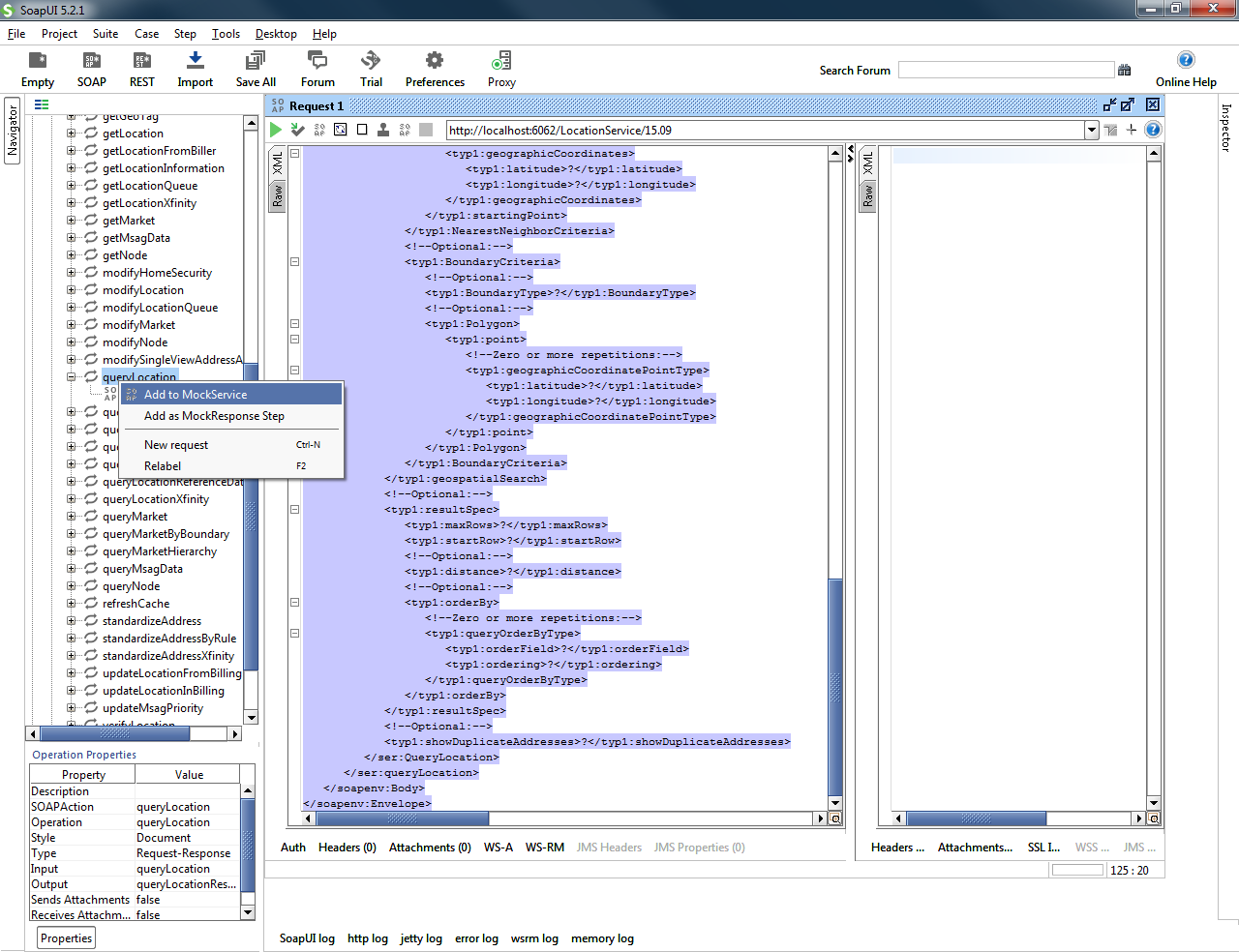
}

}

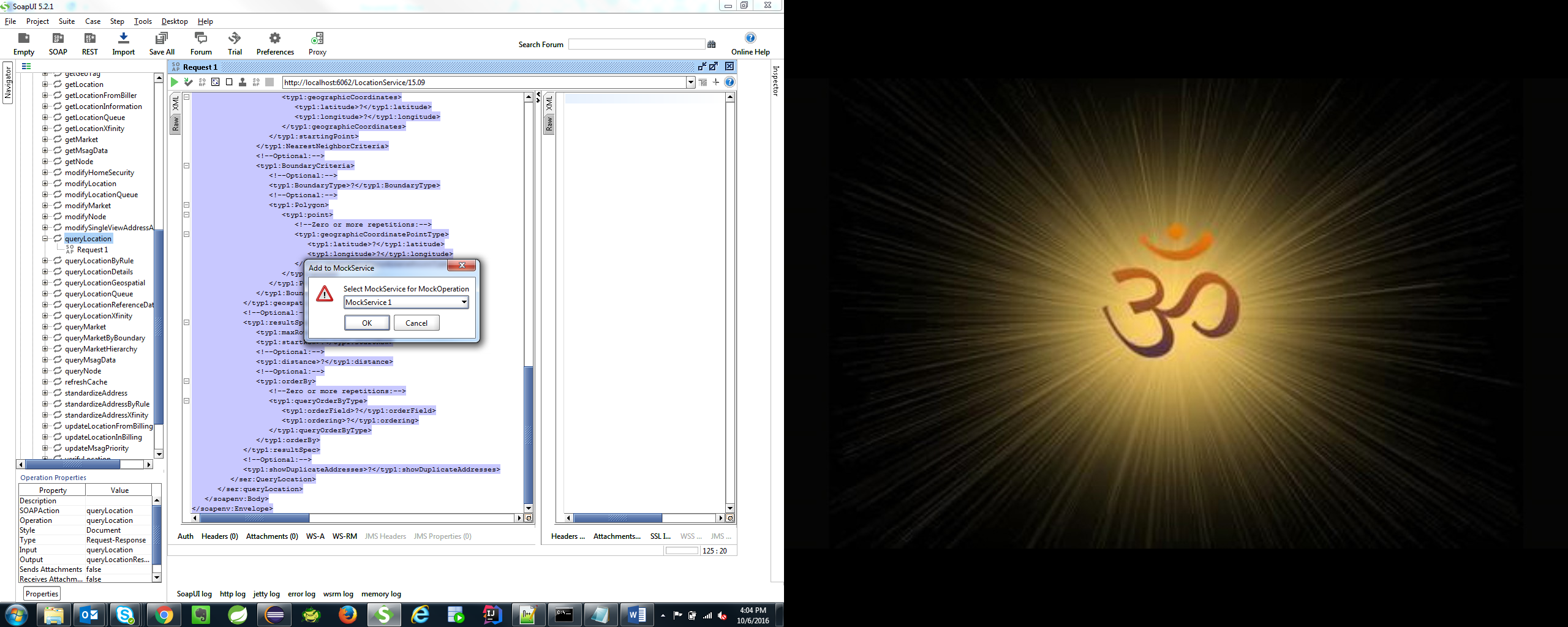
}

1. **SOAP UI –WSDL comparison process**

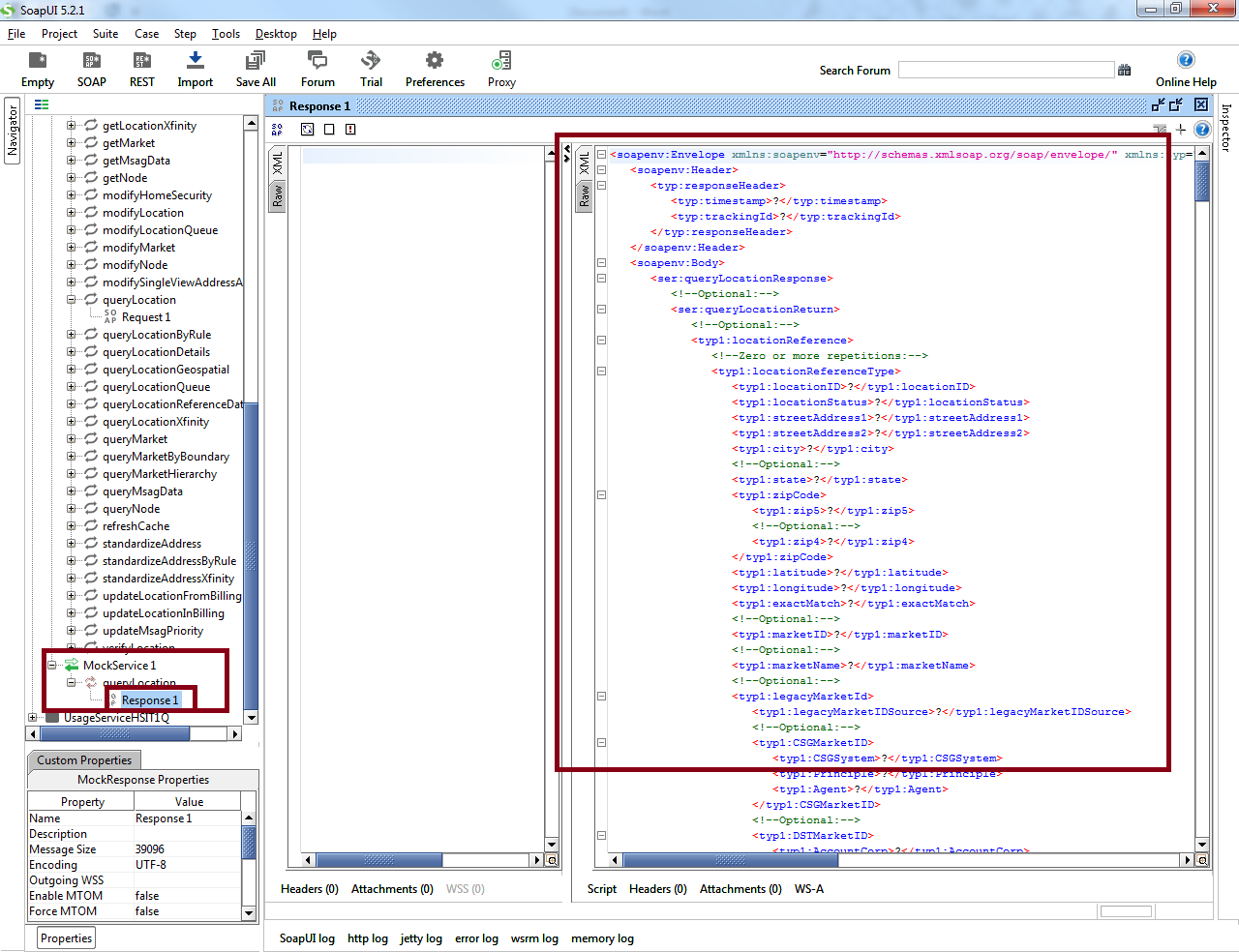
Right click on operation, select Add to MockService



Select OK in the popup



It’s open up window with mocked response object



1. **Rabbit MQ- JMS Concepts and application to Cache Refresh**

Why we need to implement JMS in ESP?

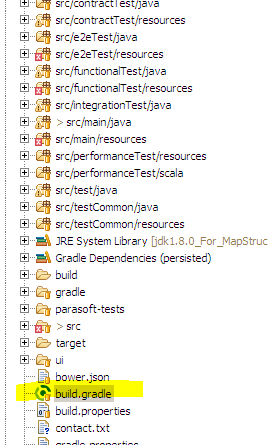
Answer: We are using Cloud in new environment with scalable feature in which we can increase number of instance of a application so for refreshing every instance cache we need to create JMS otherwise it’s hard to refresh every instance manually if instance have more than 50 or extra so avoiding manual process we just implemented Rabbit MQ.

In Comcast project MaintenanceController.java class is used for clearing cache.

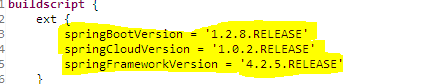
1. **Gradle- Concept + Usage**

**Gradle** is an open source build automation system that builds upon the concepts of Apache Ant and Apache Maven and introduces a Groovy-based domain-specific language (DSL) instead of the XML form used by Apache Maven of declaring the project configuration.

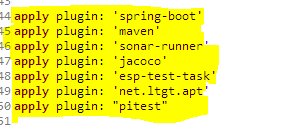
1. **Gradle file name i.e. build.gradle**



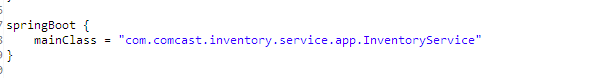
1. **Build.gradle variable declaration** :



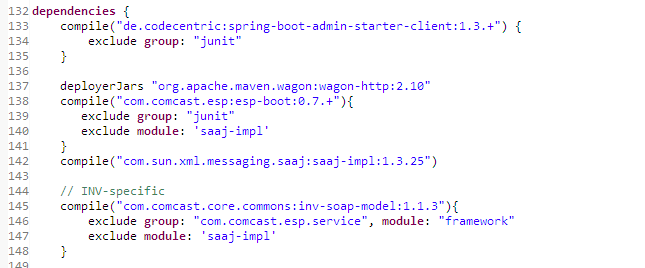
1. **applying packages :**



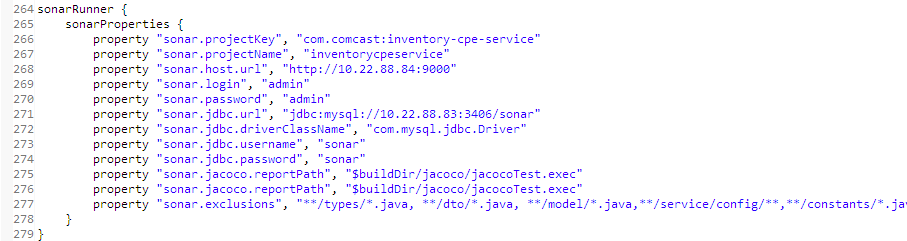
1. **Adding main class in build.gradle** :



1. **Adding Dependencies** :



1. **Adding sonar details :**



1. **Spring Cloud- Config Server and its usage**
2. Create a gradle or maven project for storing properties(\*.properties or \*.yml) file like esp-properties .
3. Create a gradle or maven project i.e ConfigServer :
4. Add these dependencies into build.gradle or pom.xml

                                                compile("org.springframework.cloud:spring-cloud-config-

server:1.0.2.RELEASE")

                                                compile("org.springframework.boot:spring-boot-starter-security")

compile("org.springframework.boot:spring-boot-starter-actuator")

1. Create a Java class file which have main method and just below annotations :

@Configuration

@EnableAutoConfiguration

@EnableConfigServer

1. For encryption/decryption we need to modify JRE so inside JRE/lib/security/ there is 2 jars i.e. US\_export\_policy.jar and local\_policy.jar which have limited strength so we have to change limited strength to Unlimited Strength.
2. For Unlimited Strength follow below steps :
3. Download Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files based on your JRE version .URL for Java 8

Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files <http://www.oracle.com/technetwork/java/javase/downloads/jce8-download-2133166.html>

1. Now rename or delete US\_export\_policy.jar and local\_policy.jar jar files from jre/lib/ security/ and unzip Unlimited Strength Jurisdiction Policy Files which you have downloaded .
2. At the end just bind modified JRE with your project.
3. Add below lines into application.properties or application.yml

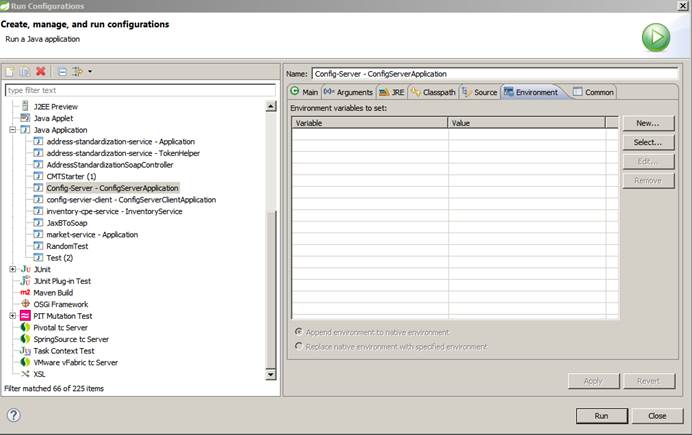
server.port= 8888(Port Number)

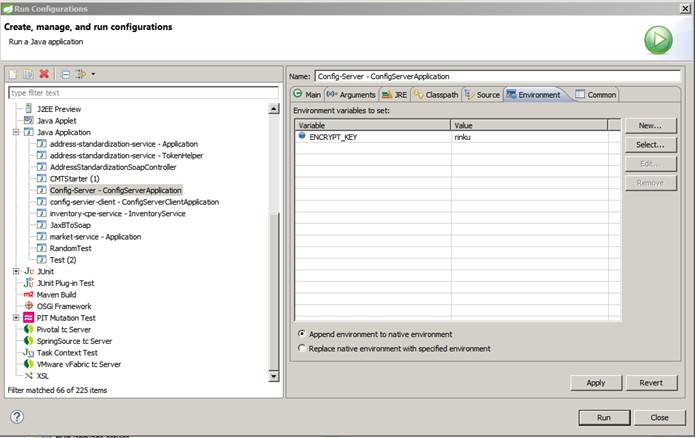
spring.cloud.config.server.git.uri={git url of your project}

security.user.password=\*\*\*\*\* -- need to login in configServer

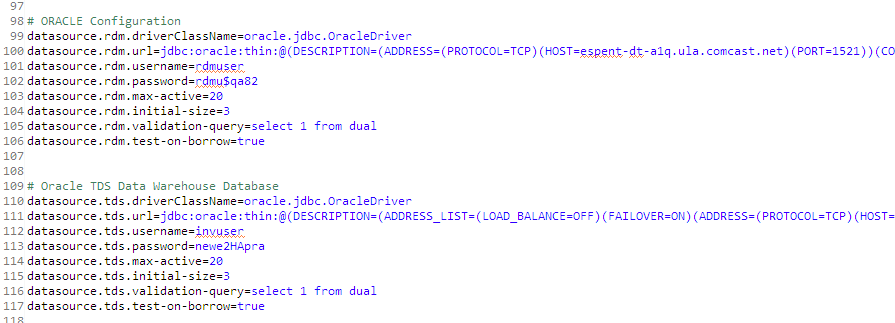
security.user.name=\*\*\*\*\* -- need to login in configServer

1. Just add ENCRYPT\_KEY into your project Environment i.e ENCRYPT\_KEY=key/encryptedkey your choice .Also adding image:

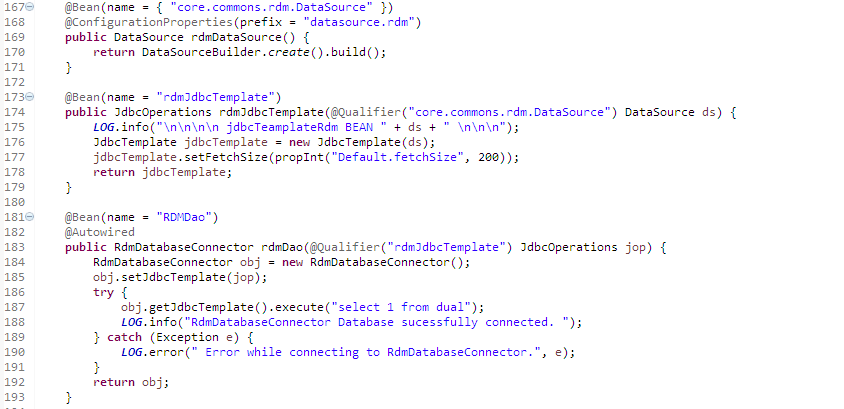




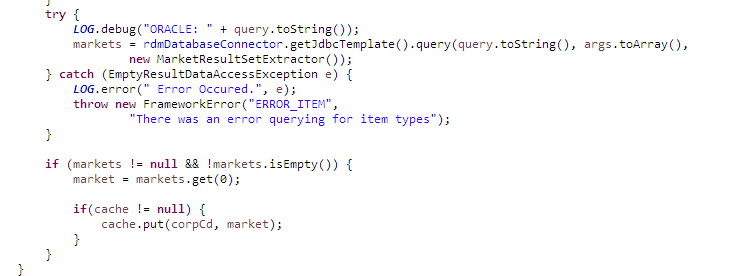
1. Now you will be able to encrypt and decrypt your password .
2. **Spring JDBC-Hibernate**
3. **JdbcTemplate with spring boot :**
4. **Add below entry in application.properties**



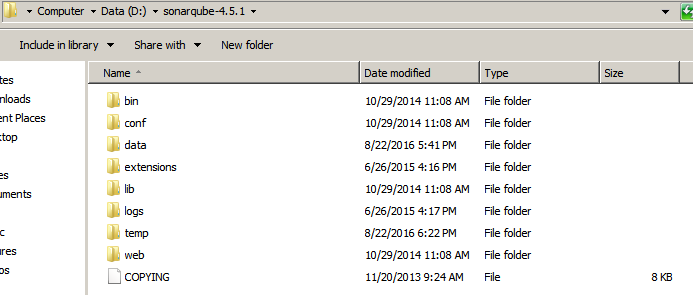
1. **Add below lines in PropertiesConfiguration.java**



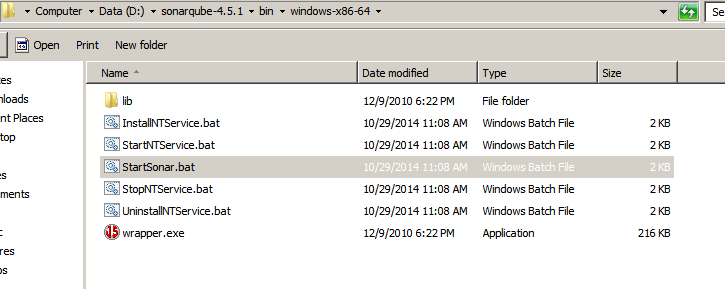
1. **Use of JdbcTemplate** :

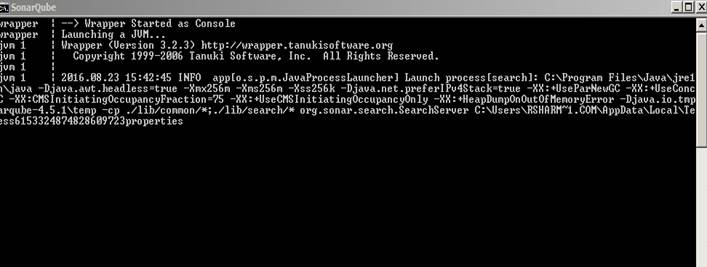


1. Sonar running steps :
2. **Download sonar-4.5.1 from  <http://downloads.sonarsource.com/sonarqube/sonarqube-4.5.1.zip> .  Paste in any directory.**

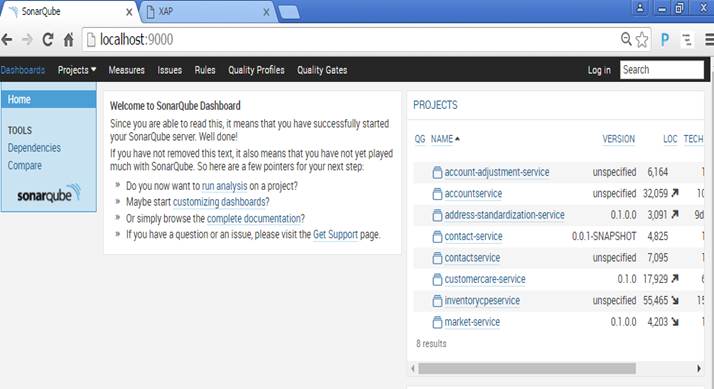


1. **Click on bin then depend on your OS like I have Windows with 64 bit then click on StartSonar.bat.**

-



1. **Now your Sonar Server will start on localhost:9000**



1. **Now move to your gradle project and open build.gradle and modify below lines in build.gradle**

**BEFORE :**

**sonarRunner {**

**sonarProperties {**

**property "sonar.projectKey", "com.comcast.esl.location:market-service"**

**property "sonar.projectName", "market-service"**

**property "sonar.host.url", "**[**http://10.22.88.84:9000**](http://10.22.88.84:9000)**"**

**property "sonar.login", "${sonar\_user}"**

**property "sonar.password", "${sonar\_password}"**

**property "sonar.jdbc.url", "jdbc:mysql://10.22.88.83:3406/sonar"**

**property "sonar.jdbc.driverClassName", "com.mysql.jdbc.Driver"**

**property "sonar.jdbc.username", "${sonar\_db\_user}"**

**property "sonar.jdbc.password", "${sonar\_db\_password}"**

**}**

**}**

**AFTER :**

**sonarRunner {**

**sonarProperties {**

**property "sonar.projectKey", "com.comcast.esl.location:market-service"**

**property "sonar.projectName", "market-service"**

**property "sonar.host.url", "**[**http://localhost:9000**](http://localhost:9000)**"**

**//property "sonar.login", "${sonar\_user}"**

**//property "sonar.password", "${sonar\_password}"**

**//property "sonar.jdbc.url", "jdbc:mysql://10.22.88.83:3406/sonar"**

**//property "sonar.jdbc.driverClassName", "com.mysql.jdbc.Driver"**

**//property "sonar.jdbc.username", "${sonar\_db\_user}"**

**//property "sonar.jdbc.password", "${sonar\_db\_password}"**

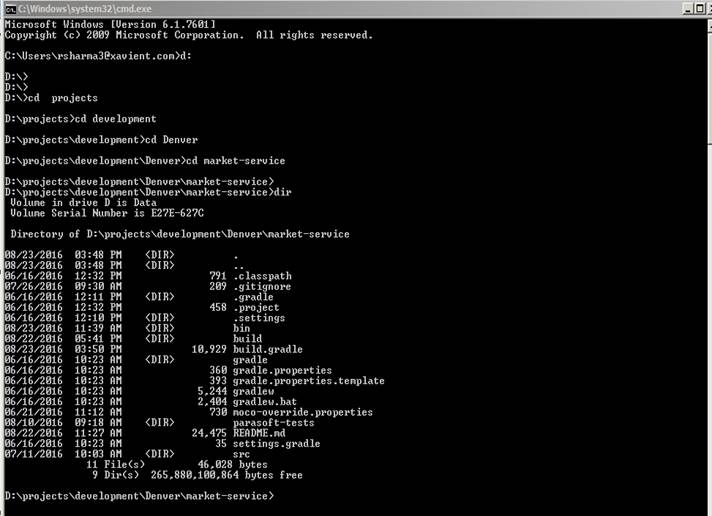
**// property "sonar.exclusions", "\*\*/types/\*\*, \*\*/domain/\*\*, \*\*/market/config/\*\*, \*\*/ElocVertexDAO.java"**

**}**

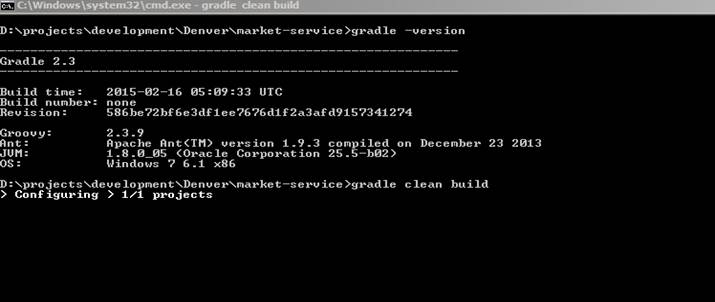
**}**

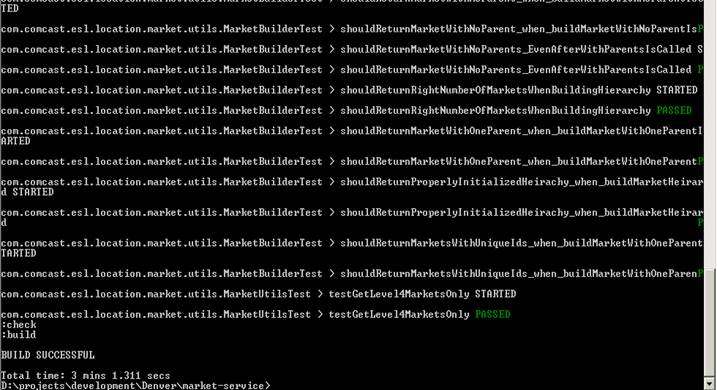


1. **Now open  command prompt  and move to your project directory .**

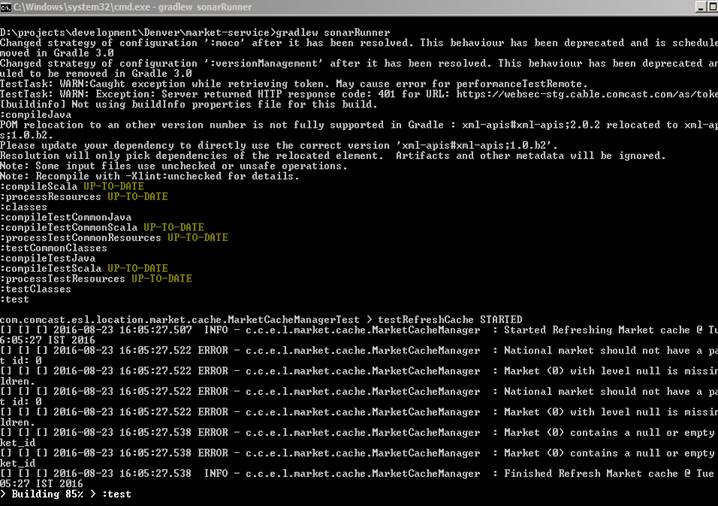


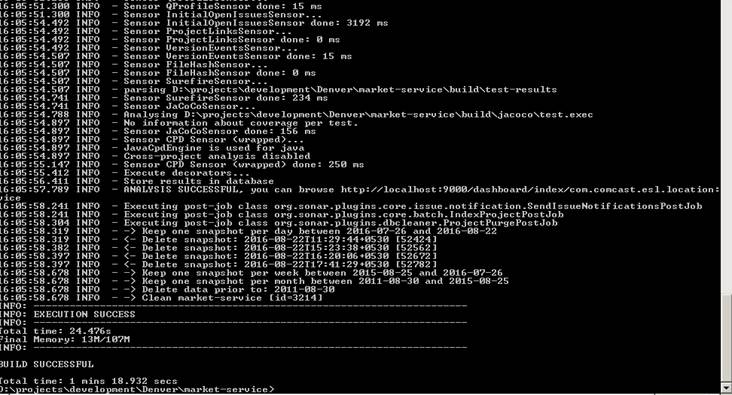
1. **Run command gradle clean build (make sure you have installed gradle in your system by command gradle -version)**



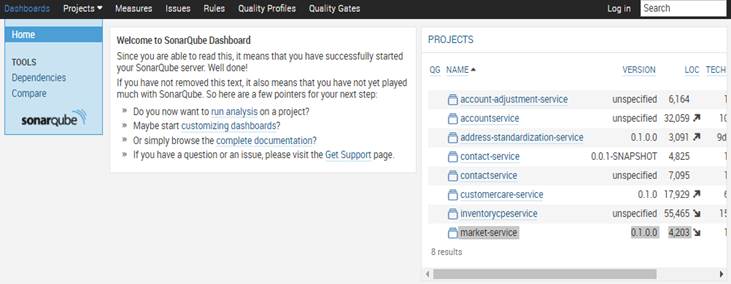


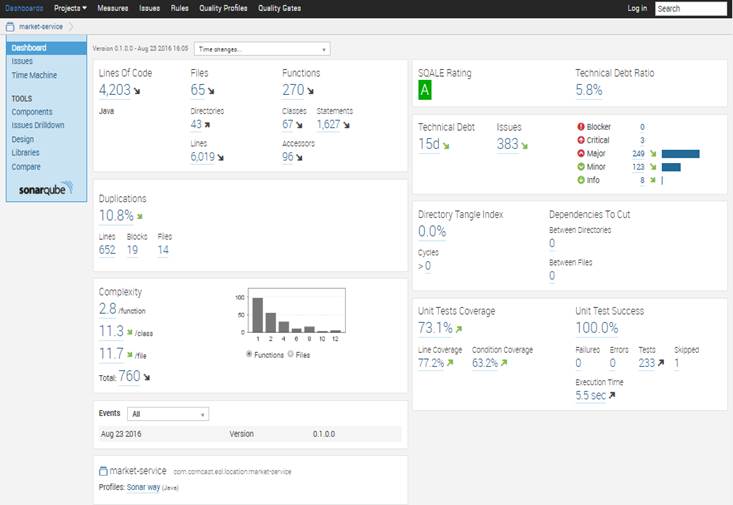
1. **After successful completion of gradle clean build just run command gradlew sonarRunner**





1. **Open localhost:9000 on browser then click on your project you will be able to find out percentage.**





**Feel free to ask any query .**

----------------------------------- END -------------------------------------------