Java

Real-time Tools

by

Mr. RAGHU

MAVEN

jar : To run any java project we must provide at least one jar to JDK(JVM). jar is a collection of .class files (or even it can contain any data like .java, .xml , .txt etc..)

All basic classes like String, Date, Thread, Exception, List etc.. are exist in rt.jar (runtime jar) given by Sun(Oracle). rt.jar will be set to project by JDK/JRE only. Programmer not required to set this jar.

Every jar is called as dependency. It means program depends on jars. Sometimes one jar may need another jar(child jar), this is called as dependency chain.

ex: Spring jars needs commons logging jar, hibernate jars needs slf4j,java-asst jar

Every jar directly or indirectly depends on rt.jar. Always parent jar version >= child jar version

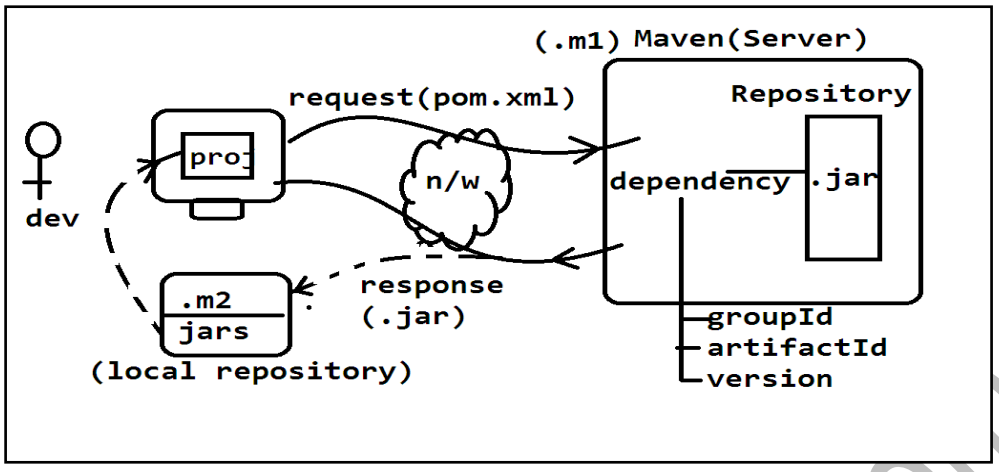
Dependency Management :

It is a concept of maintain jars of a project, maintain child jars and parent jars (dependency-chain), maintain proper version of all jars (dependency-version)

Library : It is collection of Jars.

Maven is a Tool (software) used to handle dependency-management as automated ie jars, chain jars with version management should be taken care by maven tool. It is developed by Apache. Maven is a Repository server which contains jars(dependencies) only in well-arranged format (provider based, jar name based, version based).

Every jar (depedency) contains 3 details mainly. Those are groupId (jar provider/company name) artifactId (jar name) & version (jar version). Maven maintains all jars as Library also called as Maven Library.



Maven project created in local system should make request. Request should be in XML format. That is pom.xml (pom= project object model). Maven server takes pom.xml request and returns jars as response. All jars will be copied to local repository first ie ".m2" folder.

Every jar(dependency)should contain,

groupId (providerName)
artifactId (jar name)
version (jar version).

Format looks as,

```
<dependency>
 <groupId> ___ </groupId>
 <artifactId>___ </artifactId>
 <version> ____</version>
</dependency>
```

Maven Projects :

To create Maven Project we should choose one option given below.

1. Simple Project : Basic Application
2. ArchType Project : Like web-app, framework related etc...

For any Maven project 4 basic folder are provided. Those are,

- **src/main/java**
- **src/main/resource**
- **src/test/java**
- **src/test/resource**

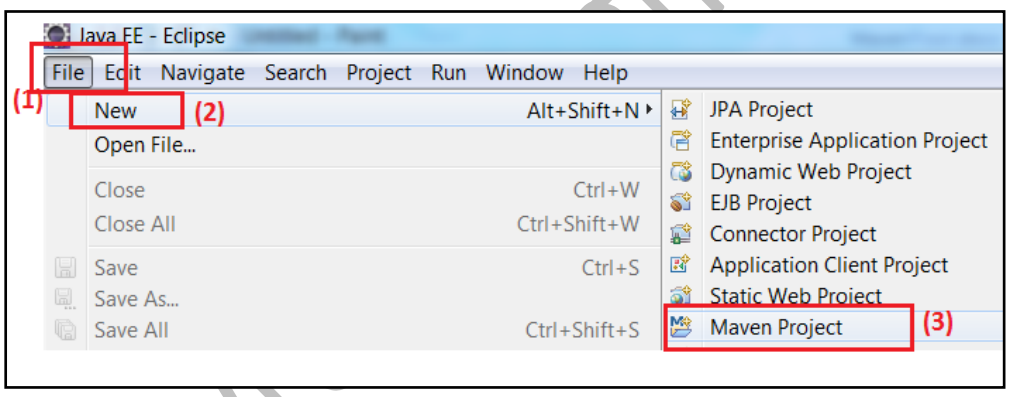
"java" folder is used to store only .java files, where "resources" are used to store non-java files like .properties, .xml, .txt, etc... "main" indicates files related to project developement. "test" indicates files related to UnitTesting.

Creating Simple Maven Project :

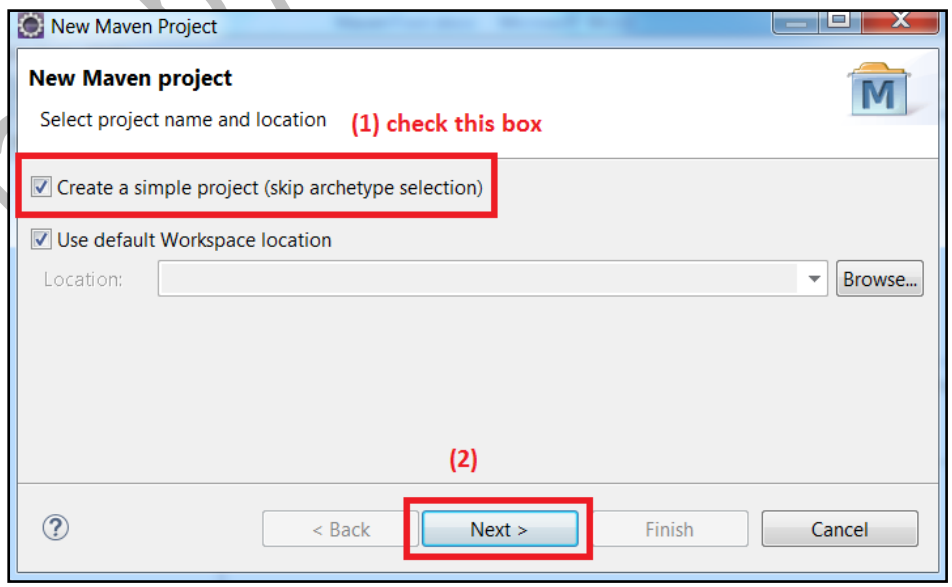
This type of projects are used in java to write JDBC,Hibernate,Spring core or any stand alone applications only. Steps to create Simple Maven Project:-

1. click on "File" menu
2. click on "New" option
3. Choose "Maven Project" option. If not exist goto "other", search using maven project word.
4. Screen looks as below. Here select checkbox "Create Simple Project".
5. on click next button, maven project needs groupId,artifactId,version details of our project.

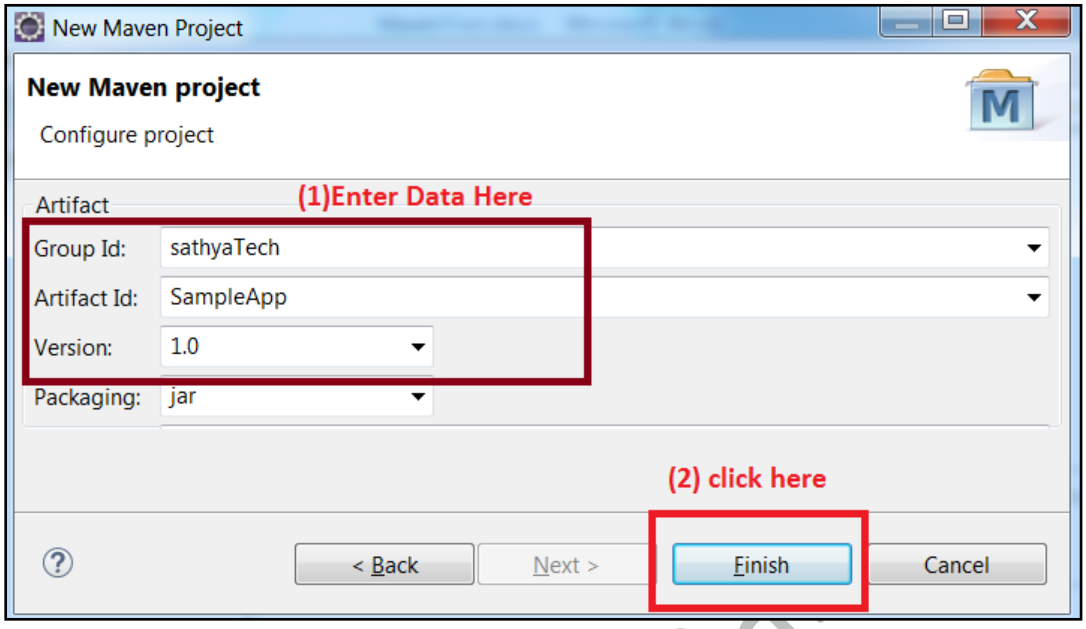
Screen : 1



Screen:2



Screen:3



At last our project also be converted/exported as .jar (java archive), .war(web archive) or .ear (enterprise archive). Here version can contain numbers, characters and symbols. (Screen looks as below)

Setup JDK/JRE in Eclipse :

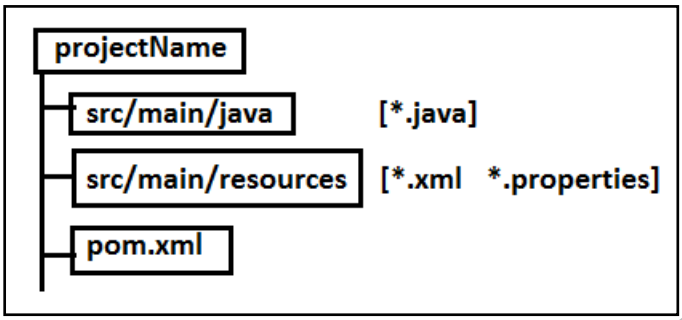
A. Window menu
B. preferences option
C. Search with "installed JRE"
D. choose Installed JRE under Java
E. click on "Add" Button to add another
F. Select "Standard VM "
G. next
H. Click on "Directory/Browse"
I. select location (ex: C:/Program Files/Java/jdk1.8)
J. finish/ok
K. finish/ok

Now Modify JRE Version to Maven

a) Right click on maven project
b) Build path option
c) Click on configure Build path..

d) Choose Library Tab
e) Select/click on "JRE System Library"
f) Click on "Edit" Button
g) Choose any one JRE
h) Finish
i) ok

Basic Folder structure for Simple Maven Project :



✓ Open pom.xml file in XML View mode. In this root tag is <project>
✓ Under this tag, add one tag
<dependencies> </dependencies>

Then file looks like

```
<project ....>
  <modelVersion>4.0.0</modelVersion>
  <groupId> ------- </groupId>
  <artifactId> ---- </artifactId>
  <version> ------- </version>
  <dependencies>



  </dependencies>
</project>
```

Here, we must specify jar details under <dependencies> tag example : for spring core programming

<dependency>

```
 <groupId>org.springframework</groupId>
 <artifactId>spring-context</artifactId>
 <version>4.3.2.RELEASE</version>
</dependency>
```

For mysql-connector jar :

```
<dependency>
 <groupId>mysql</groupId>
 <artifactId>mysql-connector-java</artifactId>
 <version>5.1.26</version>
</dependency>
```

***** User Library *****

If any Jar not exist in Maven or unable get from Maven Server (ex: created by One company or by programmer) then we can add jar to project using this concept. User Library has two steps,

1. Create Library in Workspace

2. Add Library to project.

1. Create Library in Workspace

i. Window menu
ii. Preference option
iii. Search with "User Library" word
iv. Select User Library under Java
v. Click on New button
vi. Enter Library Name (ex: MyLibs)
vii. Finish/ok
viii. Click on Library Name
ix. Click on Add External Jars Button
x. Select Jars required from folders
xi. Finish
xii. ok

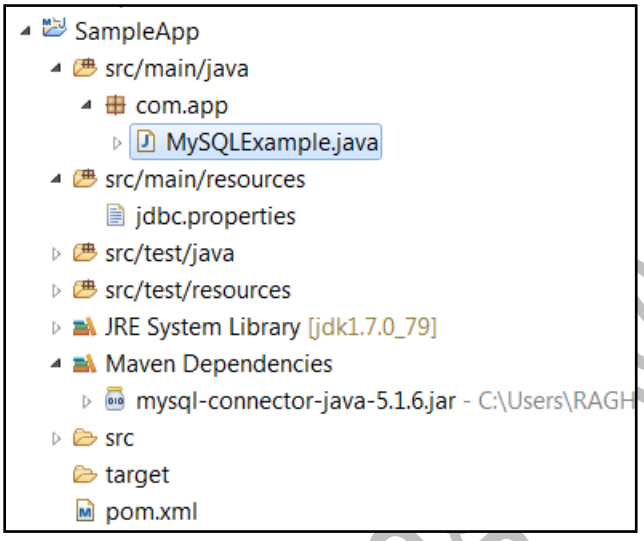
2. Add Library to project:

I. Right click on project

II. Choose "Build path"
III. Configure Build path ...
IV. Click on Library Tab.
V. Click on "Add Library" Button
VI. Select "User Library" option
VII. Next button
VIII. Choose "MyLibs"
IX. Click on Finish and ok.

EXAMPLE MAVEN APPLICATIONS

1. JDBC USING MYSQL TYPE-4 DRIVER CLASS WITH PROPERTIES FILE



pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>nareshITech</groupId>
    <artifactId>SampleApp</artifactId>
    <version>1.0</version>
    <dependencies>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>5.1.6</version>
        </dependency>
    </dependencies>
```

```
</project>
```

Step#1 Right click on src/main/resources folder , choose here new->other option. Search with "file" and choose same. Enter File name as "jdbc.properties".

```
# JDBC MySQL Connection properties

jdbc.driver=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/test
jdbc.user=root
jdbc.password=root
```

Step#2 Create one java Class "MySQLExample.java". Here defined 3 static methods to load properties file, to get DB Connection and to select data from employee table.

```
package com.app;

import java.io.InputStream;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.Properties;

/**
 * @author RAGHU
 * @version JDK 1.7
 */

public class MySQLExample {

	/**
	 * 1. Load Connection Details from Properties file
	 */
	public static Properties loadProperties(){
		Properties props=null;
		try {
			ClassLoader loader = Thread.currentThread().
```

```
                                        getContextClassLoader();
                InputStream resourceStream = loader.
                getResourceAsStream("jdbc.properties");
                props=new Properties();
                props.load(resourceStream);

        } catch (Exception e) {
                e.printStackTrace();
        }
        return props;
}

/**
 * 2. Get Connection
 *  @throws Exception
 */
public static Connection getConnection() throws Exception{

                Properties props=loadProperties();
                Class.forName(props.getProperty("jdbc.driver"));
                Connection con = DriverManager.getConnection(
                                props.getProperty("jdbc.url"),
                                props.getProperty("jdbc.user"),
                                props.getProperty("jdbc.password"));
                return con;
}
/**
 * 3. Select Data from employee table
 * @throws Exception
 */
public static void selectEmployee() throws Exception{

        Statement st = getConnection().createStatement();
        ResultSet rs = st.executeQuery("select * from employee");
        while (rs.next())    {
                        System.out.print(rs.getInt(1)+",");
                        System.out.print(rs.getString(2)+",");
                        System.out.println(rs.getDouble(3));
                }
```

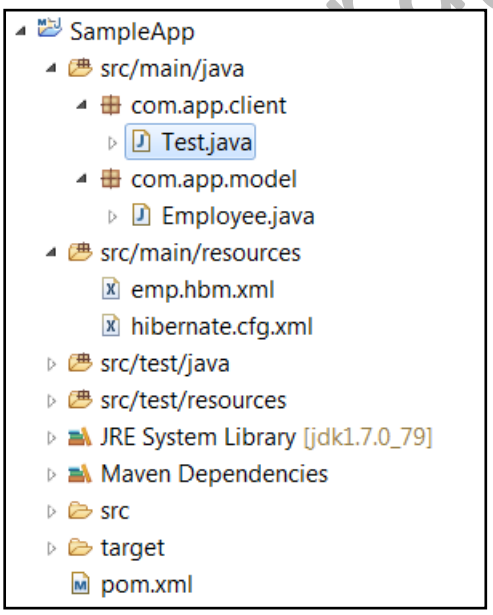
```
        }
        public static void main(String[] args) {
                try {
                        selectEmployee();
                } catch (Exception e) {
                        e.printStackTrace();
                }
        }
}
```

Create Table & insert Records:

```
employee  CREATE TABLE employee (
        empId int(11) NOT NULL,
        empName varchar(255) DEFAULT NULL,
        empSal double NOT NULL,

      ) ;
insert into employee values(10,'AA',55.23);
insert into employee values(11,'BB',66.24);
```

2. HIBERNATE EXAMPLE APPLICATION USING MYSQL DATABASE.

Folder Structure:-



pom.xml file

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <groupId>nareshITech</groupId>
        <artifactId>SampleApp</artifactId>
        <version>1.0</version>
      <dependencies>
        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-core</artifactId>
           <version>4.3.5.Final</version>
      </dependency>
      <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>5.1.6</version>
      </dependency>
     </dependencies>
</project>
```

Model class : Employee.java

```
package com.app.model;

public class Employee {

        private int empId;
        private String empName;
        private double empSal;

        //alt+shift+O (De-select All->OK)
        public Employee() {
        }
        //alt+Shift+S , R (select all) ->OK
        public int getEmpId() {
                return empId;
        }
        public void setEmpId(int empId) {
```

```
		this.empId = empId;
	}
	public String getEmpName() {
		return empName;
	}
	public void setEmpName(String empName) {
		this.empName = empName;
	}
	public double getEmpSal() {
		return empSal;
	}

	public void setEmpSal(double empSal) {
		this.empSal = empSal;
	}
	//alt+shift+s,s ->ok
	@Override
	public String toString() {
		return "Employee [empId=" + empId + ", empName=" + empName
+ ", empSal=" + empSal + "]";
	}



}
```

Mapping File : emp.hbm.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
"-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
	<class name="com.app.model.Employee" table="EMPLOYEE">
		<id name="empId" column="eid"/>
		<property name="empName" column="ename"/>
		<property name="empSal" column="esal"/>
	</class>
</hibernate-mapping>
```

configuration file hibernate.cfg.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
        <session-factory>
                <property name="hibernate.connection.driver_class">
                        com.mysql.jdbc.Driver </property>
<property
name="hibernate.connection.url">jdbc:mysql://localhost:3306/test
</property>
 <property name="hibernate.connection.user">root</property>
 <property name="hibernate.connection.password">root</property>

 <property name="hibernate.show_sql">true </property>
<property name="hibernate.dialet">org.hibernate.dialect.MySQLDialect
</property>
<property name="hibernate.hbm2ddl.auto">create</property>
<mapping resource="emp.hbm.xml" />
 </session-factory>
</hibernate-configuration>
```

Test class :

```
package com.app.client;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
import org.hibernate.cfg.Configuration;

import com.app.model.Employee;

public class Test {

        public static void main(String[] args) {
                Configuration cfg = new Configuration();
                cfg.configure(); //loads hibernate.cfg.xml file
```

```
            /*SessionFactory sf = cfg.buildSessionFactory();*/
            SessionFactory sf = cfg.buildSessionFactory(
                        new StandardServiceRegistryBuilder().
                        applySettings(cfg.getProperties()).build());

            Session ses = sf.openSession();
            Transaction tx = ses.beginTransaction();

            Employee emp=new Employee();
            emp.setEmpId(100);
            emp.setEmpName("ABC");
            emp.setEmpSal(12.36);

            ses.save(emp);

            tx.commit();
            ses.close();

            System.out.println("Record inserted..");
      }
}
```

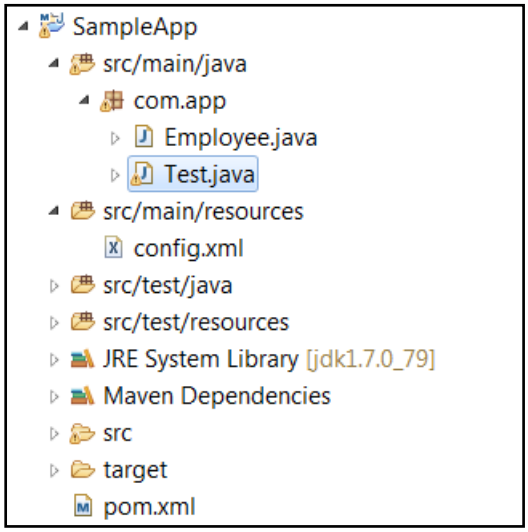
3. SPRING CORE EXAMPLE :

pom.xml file

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
      <modelVersion>4.0.0</modelVersion>
      <groupId>nareshITech</groupId>
      <artifactId>SampleApp</artifactId>
      <version>1.0</version>
      <dependencies>
            <dependency>
                  <groupId>org.springframework</groupId>
```

```
                <artifactId>spring-context</artifactId>
                <version>4.3.4.RELEASE</version>
            </dependency>
        </dependencies>
</project>
```

Folder Structure :



Spring Bean:

```
package com.app;

public class Employee {

        private int empId;
        private String empName;
        private double empSal;

        public int getEmpId() {
                return empId;
        }
        public void setEmpId(int empId) {
                this.empId = empId;
        }
        public String getEmpName() {
                return empName;
        }
        public void setEmpName(String empName) {
                this.empName = empName;
```

```
	}
	public double getEmpSal() {
		return empSal;
	}
	public void setEmpSal(double empSal) {
		this.empSal = empSal;
	}
	@Override
	public String toString() {
		return "Employee [empId=" + empId + ", empName=" + empName
+ ", empSal=" + empSal + "]";
	}

}
```

Spring Configuration File (XML File) : config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
  ">

	<bean class="com.app.Employee" name="empObj">
		<property name="empId" value="100"/>
		<property name="empName" value="ABC"/>
		<property name="empSal" value="2.36"/>

	</bean>

</beans>
```

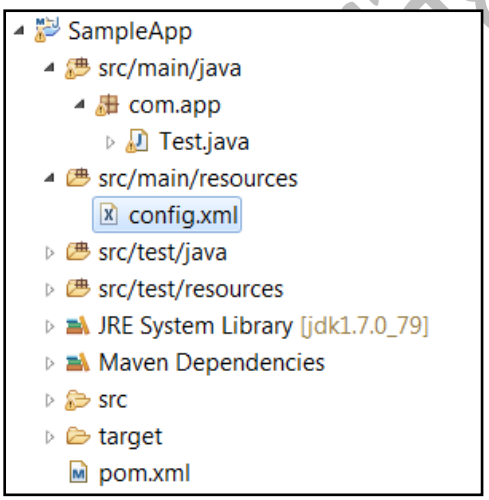
Test class:

```
package com.app;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

/**
```

```
 * @author RAGHU
 * @version JDK 1.7
 */

public class Test {

	public static void main(String[] args) {
		ApplicationContext context=new
ClassPathXmlApplicationContext("config.xml");
		Object ob=context.getBean("empObj");
		if(ob instanceof Employee){
			Employee emp=(Employee) ob;
			System.out.println(emp);
		}else{
			System.out.println("object is not employee type");
		}

	}
}
```

4. SPRING JDBC EXAMPLE :

Folder Structure:



pom.xml file:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
	<modelVersion>4.0.0</modelVersion>
	<groupId>nareshITech</groupId>
	<artifactId>SampleApp</artifactId>
	<version>1.0</version>
	<dependencies>
		<dependency>
			<groupId>org.springframework</groupId>
			<artifactId>spring-context</artifactId>
			<version>4.3.4.RELEASE</version>
		</dependency>
		<dependency>
			<groupId>org.springframework</groupId>
			<artifactId>spring-jdbc</artifactId>
			<version>4.3.4.RELEASE</version>
		</dependency>
		<dependency>
			<groupId>mysql</groupId>
			<artifactId>mysql-connector-java</artifactId>
			<version>5.1.19</version>
		</dependency>
	</dependencies>
</project>
```

Spring Configuration File (XML File): config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:p="http://www.springframework.org/schema/p"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
  ">

	<bean
class="org.springframework.jdbc.datasource.DriverManagerDataSource"
		name="dataSourceObj"
```

```
            p:driverClassName="com.mysql.jdbc.Driver"
            p:url="jdbc:mysql://localhost:3306/test"
            p:username="root"
            p:password="root"
        />

        <bean class="org.springframework.jdbc.core.JdbcTemplate"
name="jtObj"
            p:dataSource-ref="dataSourceObj"
        />

</beans>
```

Test class:

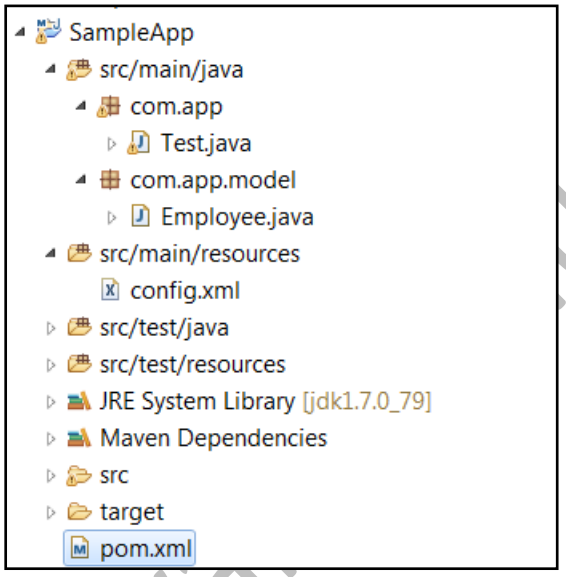
```
package com.app;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.jdbc.core.JdbcTemplate;

/**
 * @author RAGHU
 * @version JDK 1.7
 */

public class Test {

        public static void main(String[] args) {
            ApplicationContext context=new
ClassPathXmlApplicationContext("config.xml");
            JdbcTemplate jt=(JdbcTemplate) context.getBean("jtObj");
            String sql="insert into employee values(?,?,?)";
            int count=jt.update(sql, 20,"ABC",36.36);
            System.out.println(count);


        }
}
```

Create Table:

```
CREATE TABLE employee (
        empId int(11) NOT NULL,
        empName varchar(255) DEFAULT NULL,
        empSal double NOT NULL,

) ;
```

5. SPRING ORM EXAMPLE USING MYSQL DATABASE:

Folder Structure :



pom.xml file

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <groupId>nareshITech</groupId>
        <artifactId>SampleApp</artifactId>
        <version>1.0</version>
        <dependencies>
                <dependency>
```

```
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>3.2.6.RELEASE</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-orm</artifactId>
            <version>3.2.6.RELEASE</version>
        </dependency>
        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-core</artifactId>
            <version>3.6.4.Final</version>
        </dependency>
        <dependency>
            <groupId>javassist</groupId>
            <artifactId>javassist</artifactId>
            <version>3.12.1.GA</version>
        </dependency>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>5.1.19</version>
        </dependency>
    </dependencies>
</project>
```

Model class : Employee.java

```
package com.app.model;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name="employee")
public class Employee {
    @Id
    @Column(name="eid")
```

```
	private int empId;
	@Column(name="ename")
	private String empName;
	@Column(name="esal")
	private double empSal;

	public int getEmpId() {
		return empId;
	}
	public void setEmpId(int empId) {
		this.empId = empId;
	}
	public String getEmpName() {
		return empName;
	}
	public void setEmpName(String empName) {
		this.empName = empName;
	}
	public double getEmpSal() {
		return empSal;
	}
	public void setEmpSal(double empSal) {
		this.empSal = empSal;
	}
	@Override
	public String toString() {
		return "Employee [empId=" + empId + ", empName=" + empName
+ ", empSal=" + empSal + "]";
	}


}
```

Spring Configuration File (XML File) config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:p="http://www.springframework.org/schema/p"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
        xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
  ">

        <bean
class="org.springframework.jdbc.datasource.DriverManagerDataSource"
                name="dataSourceObj"
                p:driverClassName="com.mysql.jdbc.Driver"
                p:url="jdbc:mysql://localhost:3306/test"
                p:username="root"
                p:password="root"
        />

        <bean
        class="org.springframework.orm.hibernate3.annotation.AnnotationSessi
onFactoryBean"          name="sfObj">
                <property name="dataSource" ref="dataSourceObj"/>
                <property name="hibernateProperties">
                        <props>
                                <prop
key="hibernate.dialect">org.hibernate.dialect.OracleDialect</prop>
                                <prop key="hibernate.show_sql">true</prop>
                                <prop key="hibernate.hbm2ddl.auto">create</prop>
                        </props>
                </property>
                <property name="annotatedClasses">
                        <list>
                                <value>com.app.model.Employee</value>
                        </list>
                </property>
        </bean>

        <bean class="org.springframework.orm.hibernate3.HibernateTemplate"
                name="htObj" p:sessionFactory-ref="sfObj"/>

</beans>
```

Test class:

```
package com.app;
```

```
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.orm.hibernate3.HibernateTemplate;

import com.app.model.Employee;

/**
 * @author RAGHU
 * @version JDK 1.7
 */

public class Test {

	public static void main(String[] args) {
		ApplicationContext context=new
ClassPathXmlApplicationContext("config.xml");
		HibernateTemplate ht=(HibernateTemplate)
context.getBean("htObj");

		Employee emp=new Employee();
		emp.setEmpId(123);
		emp.setEmpName("ABC");
		emp.setEmpSal(12.36);
		ht.save(emp);
		System.out.println("Saved successfully..");


	}
}
```


Web Applications Using Maven :

To create Web Applications using maven,we must use "maven-archtype-webapp" option. To Run this types of applications we need server (any one server. ex: Tomcat) in workspace configured.

To configure new server in workspace, steps are

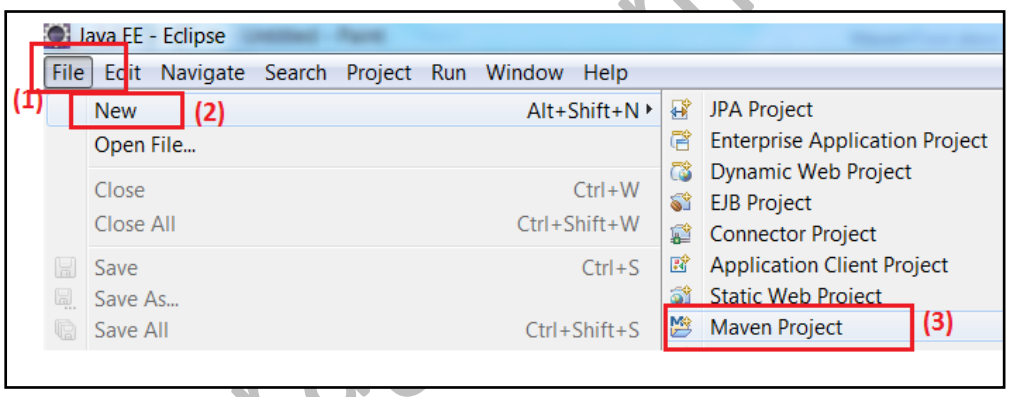
i. Right click Server (tab)
ii. choose new

iii. server (option)
iv. select server ex: Apache / Tomcat 6.0
v. click on next
vi. Browse for location ex: "C:/Program Files/Apache Software Foundation/Tomcat 6.0"
vii. Click on finish.

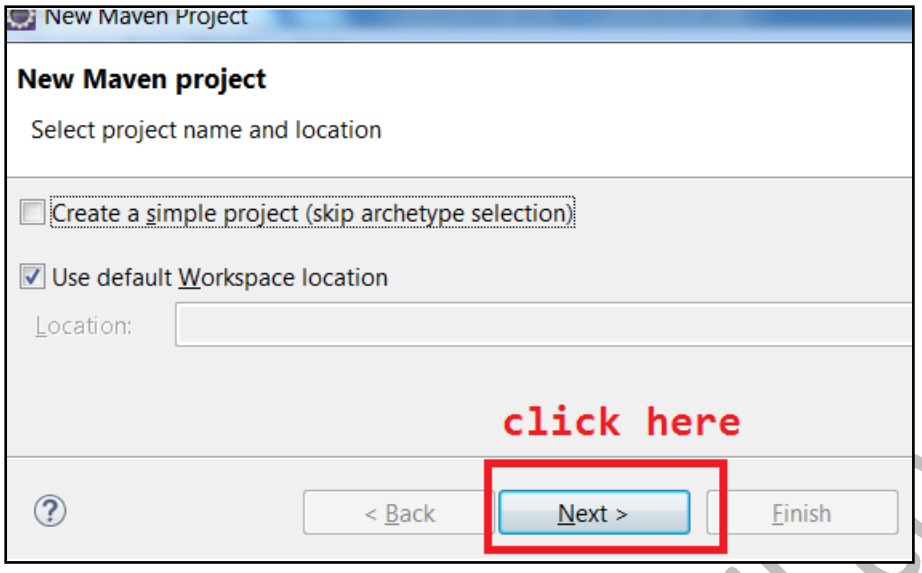
Creating maven web project:

Step#1 create project

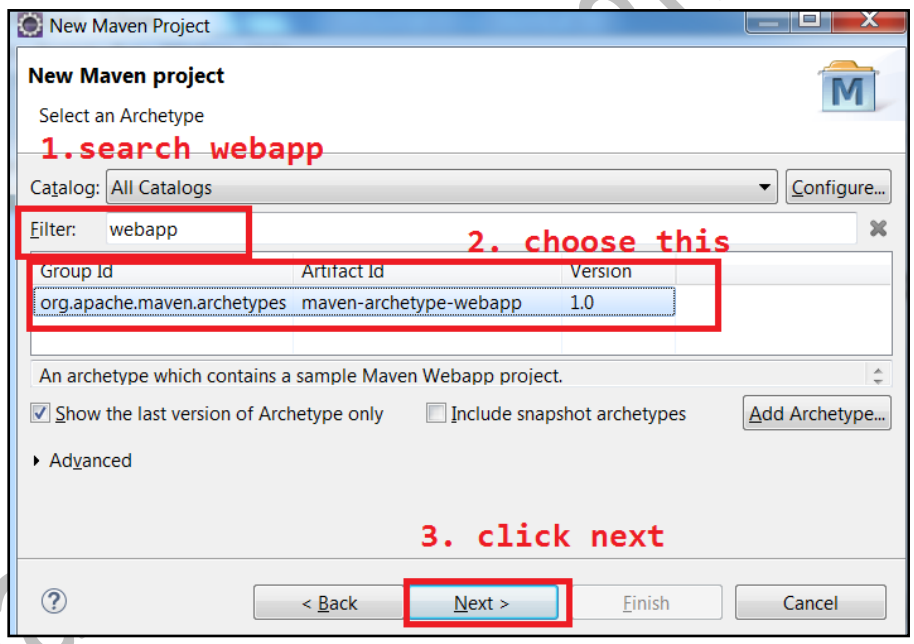
1. Click on File menu
2. choose new option
3. select Maven Project



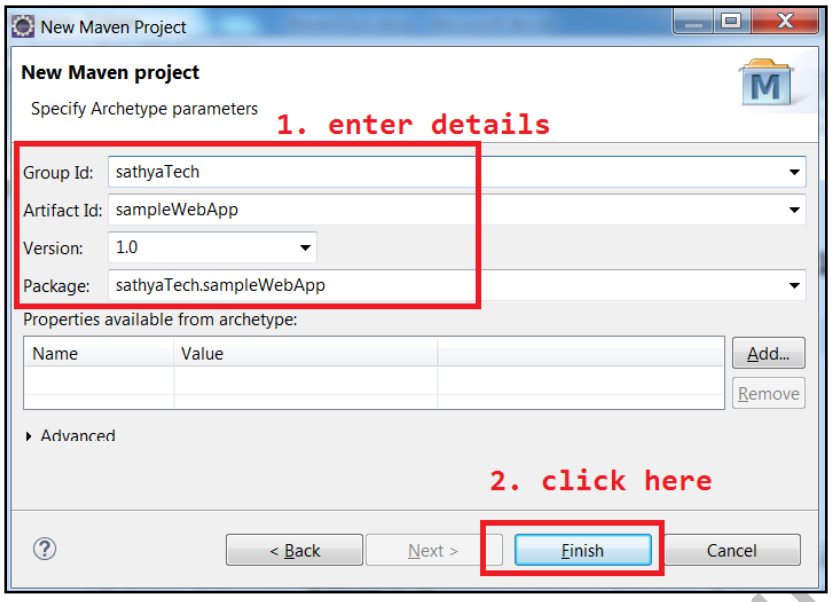
4. Do not select check box "create simple Maven project"
5. Click next



6. Select maven-archtype-webapp option by searching in filter as 'webapp'.



7. Enter details like GroupId, artifactId, version
8. Click on finish



Step#2 Configure JDK/JRE and Tomcat server

1. Right click on project
2. Choose build path
3. Click on configure build path ...
4. Select "Library" tab.
5. Click on "JRE System Library" option
6. Click Edit button
7. Choose update JRE
8. Click on "Add Library "
9. Choose "Server Runtime"
10. Click on Next Button
11. Select Server "Apache Tomcat"
12. Click on Finish
13. Click on Finish/OK.

NEED OF LOG4J

BUG/Problem:- After Application is developed then it will be handover to End Client/Customer. At the time of Running Application some problems may occur while processing a request. In programming concept it is called as Exception.

These problems try to identify at the time of development to avoid few. Some will be handled once they are occurred. Those are analyzed and solved by developer. Give priority to Avoid problems if not handle the problems.

To avoid problems concepts are : 1. Validations 2. Standard Coding

To handle the problems concepts are: 1. Error pages **2.Log4J**

1. Validations : At UI page like Register, Data Edit, Login, Feedback form, Comment etc.. at these levels we must use Java Script or equal Programming to avoid invalid input. Example:-

Register Page (JSP)
ID [_____] Only numarics [0-9]
Name [_____] only characters [a-z]
Email [_____] Pattern __@__.__ format

Write Scripting code for above checking.

_____

2. Standard Coding : If we use coding standards we can reduce [30-40]% problems in application.

Examples : -

i) String s=(...) input is provided at run time
int len=s.length();
above code is converted to...
String s=....;
if(s!=null) int len=s.length();

else sysout("No Valid Data");

ii) **int arr[]={....};**
sysout(arr[4]);
may throw ArrayIndexOutOfBounds Exception above code is converted to...

```
int arr[]={....};
if(arr.length > 4){
 sysout(arr[4]);
}else {
 sysout("Array Size is less");
}
```

iii) **Object ob=ses.get(T.class,id);** (T=className provided at runtime)
Address a=(Address)ob;
it may throw ClassCastException to avoid this

```
if(ob instanceof Address){
  Address a=(Address)ob;
} else {   sysout("ob is not address");    }
```

iv) List<Employee> has Employee objects read 1st index employee and find that employee object empName length

code

```
List<Employee> empList=....
Employee e=list.get(0);
String empName=e.getEmpName();
int len=empName.length();
```

above code can be written as.....

```
if(empList!=null && empList.size>0 && empList.get(0)!=null ){
  Employee e=list.get(0);
  if(e!=null && e.getEmpName()!=null ) {
    int len=e.getEmpName().length();
  }
}
```

3. Error Pages in Web Applications:- If server has encountered any problem then it will throw An equal HTTP Error. Example 404-Page Not Found, 500-Internal server Error, 400-Syntax Error, 405-Method Not Allowed, 401-UnAuthorized etc...

To show one simple message to end user we must define JSP/HTML page and make them as isErrorPage="true" and configure in **web.xml** as

```
<error-page>
  <error-code>404</error-code>
   <location>/error.jsp</location>
</error-page>
<error-page>
 <exception-type>java.lang.NullPoniterException</exception-type>
 <location>/showError.jsp</location>
</error-page>
```

Create on "error.jsp" ex:

```
<%@ page ... isErrorPage="true"%>
<html><body> Welcome to Error Page ..Wait 5 sec to redirect to Home Page..
<%response.setHeader("Refresh", "5;index.jsp"); %> </body> </html>
```


Log4J

1. It is a Tracing or Logging Tool used in Specially in Production Environment. It is used to find success messages, information, warnings, errors in application while using it.
2. By Default any Message/Error will be printed on Console, which is temporary location, it can show only few lines like last 100 lines.
3. To see all problems from beginning to till date use Log4j concept.

4. Log4J can write problems/error to File(.log), Database, Email, Network etc..
5. Log4J provides Error Details like Exception type, class and method with line number it occured, Date and time also other information ..
6. Log4J also supports writing errors to Console also.
7. Using System.out.println prints only on console, but not to file or any other memory.

Log4J has 3 components:

1. Logger (LOG) Object: This object must be created inside the class as a instance variable. It is used to enable Log4J service to current class. If this object is not created in the class then Log4J concpet will not applicable(will not work)for that class It has 5 priority methods with names (along with order)

Order	method	Name
1	debug(obj)	DEBUG
2	info(obj)	INFO
3	warn(obj)	WARN
4	error(obj)	ERROR
5	fatal(obj)	FATAL
-NA-	-NA-	OFF

NA : Not Applicable

a. debug(msg) : It prints a message with data. It is used to print a final result of process. Like EmpId after saved is : 2362.
b. info(msg) : It is used to print a simple message. Like process state-I done, if block end. Email sent etc..
c. warn(msg): It is used to print warning messages. Like Collection is not with Generic, local variable not used, resource not closed etc...
d. error(msg): It is used to print Exceptions like NullPointer, ArrayIndex, SQLException etc..

e. fatal(mgs) : It indicates very high level problem. Like Server/DB Down, Connection timeout, Network broken, Class Not Found etc...

OFF is used to disable Log4J concept in application. Log4J code need to be deleted.

---

2. Appender : It will provide details for "Where to print Message?". Means in what type of memories, messages must be stored. To write Logger Statements to

i. File use FileAppender
ii. Database use JdbcAppender
iii. Email use SmtpAppender
iv. Console use ConsoleAppender
v. Network use Ftp(Telnet)Appender

In one Application we can use more than one appender also.

---

3. Layout : It provide the format of message to be printed. Possible layouts are:

i. Simple Layout : Print message as it is
ii. HTML Layout : Print message in HTML format(<html><body>....)
iii. XML Layout: Print message in XML format (<Errors><Type>..<Message>..)
iv. Pattern Layout : Prints messages in given pattern. example pattern: Date-Time / Line Number : Class-method :- Message

In development mostly used Appender is FileAppender and Layout is Pattern Layout.

Log4J Example Programming:-

Step#1 Create Maven Project

I. Click on File Menu
II. Choose "new" option

III. Choose Maven Project
IV. Select "Create Simple Project" checkbox
V. Click on "Next" Button
VI. Enter GroupId, ArtifactId, version details
VII. Click on "finish" button

Step#2 Specify Log4J dependency in pom.xml

```
<dependency>
  <groupId>log4j</groupId>
  <artifactId>log4j</artifactId>
  <version>1.2.17</version>
</dependency>
```

Step#3 Create one class under "src/main/java" and write Log4J code.

ex: code

```
package com.app;
import org.apache.log4j.Appender;
import org.apache.log4j.ConsoleAppender;
import org.apache.log4j.Layout;
import org.apache.log4j.Logger;
import org.apache.log4j.SimpleLayout;

public class Product {
        private static Logger log=Logger.getLogger(Product.class);
        public static void main(String[] args) {
                Layout layout=new SimpleLayout();
                Appender ap=new ConsoleAppender(layout);
                log.addAppender(ap);
                log.info("Hello");
        }
}
```

Run application "ctrl+F11" to see output.

Note:

1. To Create "Logger" class object to current class use static factory method getLogger(_.class). It returns Logger Object and enable the Log4J to current class.

2. Create a Layout(Abstract Class) object using it's implemented classes, list given below

a. SimplLayout (C)
b. XMLLayout (C)
c. HTMLLayout(C)
d. PatternLayout(C)

3. Create Appender(I) object using it's implemented classes and also provide layout details to Appender object.

a. ConsoleAppender (C)
b. FileAppender (C)
c. JdbcAppender (C)
d. SmtpAppender (C)

4. At last Appender object must be added to Logger Object then only it is considered, by using log.addAppender() method.

5. If No appender is provided to Logger object then Log4J throws WARNING as log4j:WARN No appenders could be found for logger (com.app.Product).

Layouts: Layout Defines Format of Message to be displayed on UI/DB/Console. Possible Layout classes are listed below with description.

1. SimpleLayout : This class prints message as it is provided in log methods.

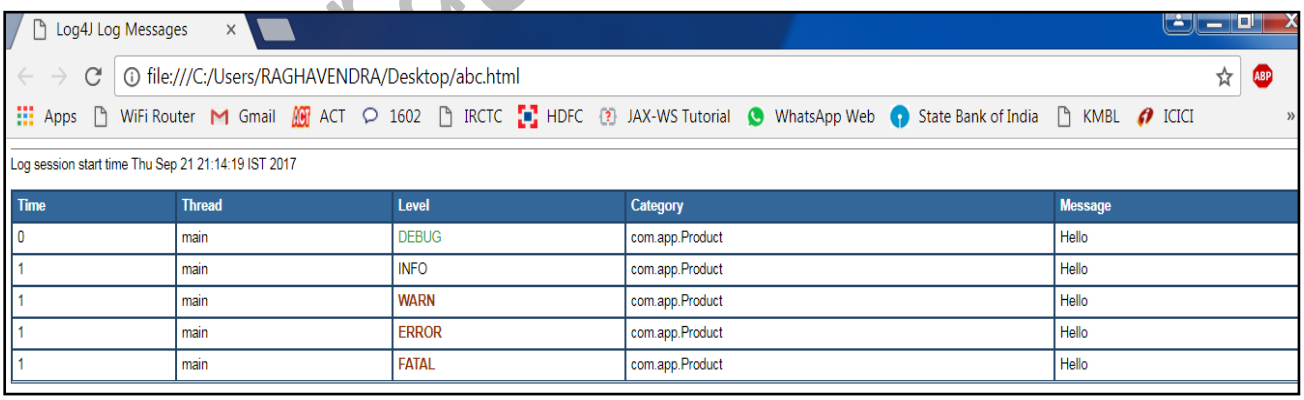
2. HTMLLayout : This class provides message in HTML format. To see full output of this. Copy output to .html file

Example Code:

```
package com.app;
import org.apache.log4j.Appender;
```

```
import org.apache.log4j.ConsoleAppender;
import org.apache.log4j.HTMLLayout;
import org.apache.log4j.Layout;
import org.apache.log4j.Logger;


public class Product {
	private static Logger log=Logger.getLogger(Product.class);
	public static void main(String[] args) {
		Layout layout=new HTMLLayout();
		Appender ap=new ConsoleAppender(layout);
		log.addAppender(ap);

		log.debug("Hello");
		log.info("Hello");
		log.warn("Hello");
		log.error("Hello");
		log.fatal("Hello");

	}
}
```

Example: screen



Time	Thread	Level	Category	Message
0	main	DEBUG	com.app.Product	Hello
1	main	INFO	com.app.Product	Hello
1	main	WARN	com.app.Product	Hello
1	main	ERROR	com.app.Product	Hello
1	main	FATAL	com.app.Product	Hello

3. XMLLayout: It prints log messages in XML format. In above code modify Layout as XMLLayout and run as same.

Example Code:

```
package com.app;
```

```
import org.apache.log4j.Appender;
import org.apache.log4j.ConsoleAppender;
import org.apache.log4j.Layout;
import org.apache.log4j.Logger;
import org.apache.log4j.xml.XMLLayout;


public class Product {
	private static Logger log=Logger.getLogger(Product.class);
	public static void main(String[] args) {
		Layout layout=new XMLLayout();
		Appender ap=new ConsoleAppender(layout);
		log.addAppender(ap);

		log.debug("Hello");
		log.info("Hello");
		log.warn("Hello");
		log.error("Hello");
		log.fatal("Hello");

	}
}
```

Output Looks as :

```
<log4j:event logger="com.app.Product" timestamp="1506008834191"
level="DEBUG" thread="main">
<log4j:message><![CDATA[Hello]]></log4j:message>
</log4j:event>

<log4j:event logger="com.app.Product" timestamp="1506008834192"
level="INFO" thread="main">
<log4j:message><![CDATA[Hello]]></log4j:message>
</log4j:event>

<log4j:event logger="com.app.Product" timestamp="1506008834192"
level="WARN" thread="main">
<log4j:message><![CDATA[Hello]]></log4j:message>
</log4j:event>
```

```
<log4j:event logger="com.app.Product" timestamp="1506008834192"
level="ERROR" thread="main">
<log4j:message><![CDATA[Hello]]></log4j:message>
</log4j:event>

<log4j:event logger="com.app.Product" timestamp="1506008834192"
level="FATAL" thread="main">
<log4j:message><![CDATA[Hello]]></log4j:message>
</log4j:event>
```

4. PatternLayout : This class provides output pattern for a message that contains date, time, class, method, line number, thread name, message etc..

Observe Some example patterns given below, for complete details click on below link

(https://logging.apache.org/log4j/1.2/apidocs/org/apache/log4j/PatternLayout.html)

a)Date&Time pattern

- %d = date and time

examples:

a) %d
b) %d {dd-MMM-yy hh:mm:ss SSS}
c) %d {dd-MM-yy hh:mm:ss SSS}
d) %d {HH:mm:ss}
e) Here meaning of every word used
f) in date pattern is,
g) dd = date
h) MMM= Month Name
i) MM = Month number
j) yy = Year last two digitis
k) yyy = Year in 4 digits
l) hh = Hours in 12 format
m) HH = Hours in 24 format

n) mm = Minutes
o) ss = Secs
p) SSS = MillSecs

- %C = Class Name
- %M = Method Name
- %m = Message
- %p = Priority method name(DEBUG,INFO..)
- %L = Line Number
- %l = Line number with Link
- %n = New Line(next line)
- %r = time in milli sec.
- %% = To print one '%' symbol.
- we can also use symbols like - [] , /

One Example Pattern with all matching "%d-%C[%M] : {%p}=%m<%L> %n "

Appender:- An appender provides the destination of log message. Here example appenders are

a) ConsoleAppender
b) FileAppender
c) JdbcAppender
d) SmtpAppender

Example of FileAppender:-

```
package com.app;

import org.apache.log4j.Appender;
import org.apache.log4j.FileAppender;
import org.apache.log4j.HTMLLayout;
import org.apache.log4j.Layout;
import org.apache.log4j.Logger;


public class Product {
	private static Logger log=Logger.getLogger(Product.class);
```

```
    public static void main(String[] s)       throws Exception {
            Layout layout=new HTMLLayout();
            Appender ap=new FileAppender(layout,"myapp.log");
            log.addAppender(ap);

            log.debug("Hello");
            log.info("Hello");
            log.warn("Hello");
            log.error("Hello");
            log.fatal("Hello");
    }
}
```

log4j.properties

log4j.properties file: This file is used to specify all the configuration details of Log4J. Especially like Appender Details and Layout Details with Patterns and also root Logger details. This File contains details in key=value format (.properties). Data will be shown in below order like

1. rootLogger
2. appenders
3. layouts

a) In this file (log4j.properties) use symbol '#' to indicates comments.
b) We can specify multiple appenders in log4j.properties file Like 2 File Appenders, one JdbcAppender, One SmtpAppender etc..
c) Every Appender must be connected with layout
d) Appender name must be define before use at rootLogger level.
e) Appender name can be anything ex: abc,hello,sysout,file,db,email etc..
f) log4j.properties file must be created under src folder (normal project) or src/main/resource folder (maven project).
g) Make rootLogger = OFF to disable Log4J completely without deleting code in any class or properties file
h) log4j.properties file will be auto detected by Log4J tool. No special coding is required.

Example #1 Console Appender

Create one properties file under "src/main/resources" with name "log4j.properties"

I. Right click on "src/main/resources"
II. Choose "new"
III. Select "other.."
IV. Search and select "file" for option
V. Click on next
VI. Enter file name and click finish.

Example : log4j.properties (code)

```
# Root logger details
log4j.rootLogger=DEBUG,stdout

# Redirect log messages to console
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.Target=System.out
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{yyyy-MM-dd} %p %c:%L - %m%n
```

Test class:-

```
package com.app;
import org.apache.log4j.Logger;
public class Product {
	private static Logger log=Logger.getLogger(Product.class);
	  public static void main(String[] srs){
		log.debug("Hello");
		log.info("Hello");
		log.warn("Hello");
		log.error("Hello");
		log.fatal("Hello");

	}
```

```
}
```

output:

```
2017-09-21 21:47:59 DEBUG com.app.Product:6 - Hello
2017-09-21 21:47:59 INFO com.app.Product:7 - Hello
2017-09-21 21:47:59 WARN com.app.Product:8 - Hello
2017-09-21 21:47:59 ERROR com.app.Product:9 - Hello
2017-09-21 21:47:59 FATAL com.app.Product:10 - Hello
```

Using of Logger methods:

1. debug() : to specify a message with value (message with result after operation)
2. info() : to indicate current state (block end, method started , service finished, email sent etc..)
3. warn() : to provide warning message (Variable not used, List has no generic, connection not closed)..
4. error/fatal : used in catch blocks. Here error - used for unchecked exception and fatal - checked exception in general

example :- spring controller

```
package com.app.controller;
@Controller
public class HomeController {
	private static Logger log=Logger.getLogger(HomeController.class);

	@RequestMapping("/url")
	public String saveEmp(...){
		log.info("Save Method started");
		try{
			log.info("Before save employee..");
			int empId=service.save(emp);
			log.debug("emp saved with id :"+empId);
		}catch(Exception e){
			log.error(e); // log.fatal(e);
```

```
            }
            log.warn("EMPID not used in program");
            log.info("save service is at end");
            return "Home";
        }
}
```

Log4j Examples

pom.xml (to run all below examples)

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <groupId>nareshItech</groupId>
        <artifactId>log4jexample</artifactId>
        <version>1.0</version>
        <dependencies>
                <dependency>
                        <groupId>log4j</groupId>
                        <artifactId>log4j</artifactId>
                        <version>1.2.17</version>
                </dependency>
                <dependency>
                        <groupId>mysql</groupId>
                        <artifactId>mysql-connector-java</artifactId>
                        <version>5.1.6</version>
                </dependency>
                 <dependency>
                                <groupId>javax.mail</groupId>
                                <artifactId>mail</artifactId>
                         <version>1.4</version>
                </dependency>
        </dependencies>

</project>
```

Example #1 Console Appender Example

Step#1 : Define one Test class with example code with log methods.

```
package com.app;
import org.apache.log4j.Logger;
public class Product {
        private static Logger log=Logger.getLogger(Product.class);
          public static void main(String[] srs){
                log.debug("Hello");
                log.info("Hello");
                log.warn("Hello");
                log.error("Hello");
                log.fatal("Hello");

        }
}
```

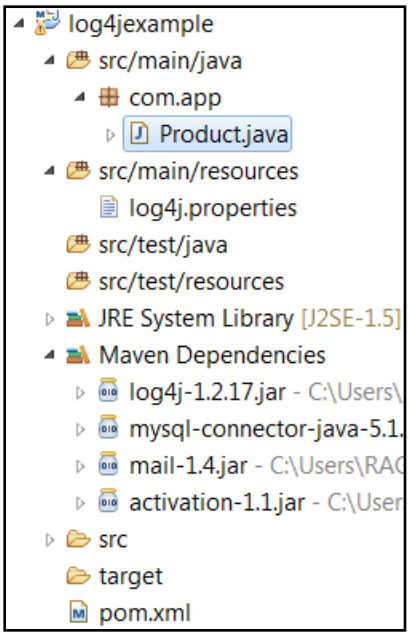
Step#2 Create log4j.properties file with below code

log4j.propertie:

```
# Root logger option
log4j.rootLogger=DEBUG, stdout
# Redirect log messages to console
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.Target=System.out
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{yyyy-MM-dd } %p
%c:%L - %m%n
```

Step#3 Run above Test class Product.java (ctrl+F11) to see output.

Folder Structure:-



Example#2 File Appender Example

Step#1 : Define one Test class with example code with log methods.

```
package com.app;
import org.apache.log4j.Logger;
public class Product {
        private static Logger log=Logger.getLogger(Product.class);
          public static void main(String[] srs){
                  log.debug("Hello");
                  log.info("Hello");
                  log.warn("Hello");
                  log.error("Hello");
                  log.fatal("Hello");

        }
}
```

Step#2 Create log4j.properties file with below code

log4j.properties

```
# Root logger option
log4j.rootLogger=DEBUG, file
```

```
# Redirect log messages to a log file
log4j.appender.file=org.apache.log4j.FileAppender
log4j.appender.file.File=logs/myapp.log
log4j.appender.file.layout=org.apache.log4j.PatternLayout
log4j.appender.file.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss}
%p %c:%L - %m%n
```

Step#3 Run above Test class Product.java (ctrl+F11) to see output.

Example#3 JDBC Appender Example

Step#1 : Define one Test class with example code with log methods.

```
package com.app;
import org.apache.log4j.Logger;
public class Product {
        private static Logger log=Logger.getLogger(Product.class);
          public static void main(String[] srs){
                  log.debug("Hello");
                  log.info("Hello");
                  log.warn("Hello");
                  log.error("Hello");
                  log.fatal("Hello");

        }
}
```

Step#2 Create this table before you run example, all the logs will be stored in below table

```
CREATE TABLE LOGS (
        METHOD VARCHAR(20)  NOT NULL,
        DATED   DATETIME                NOT NULL,
        LOGGER  VARCHAR(50)    NOT NULL,
        LEVEL   VARCHAR(10)     NOT NULL,
        MESSAGE VARCHAR(1000)       NOT NULL
);
```

Step#3 Create log4j.properties file with below code

log4j.properties

```
# Root logger option
log4j.rootLogger=DEBUG, db

# Define the Jdbc appender
log4j.appender.db=org.apache.log4j.jdbc.JDBCAppender
log4j.appender.db.driver=com.mysql.jdbc.Driver
log4j.appender.db.URL=jdbc:mysql://localhost:3306/test
log4j.appender.db.user=root
log4j.appender.db.password=root
log4j.appender.db.layout=org.apache.log4j.PatternLayout
log4j.appender.db.sql=INSERT INTO LOGS VALUES ('%M', now()
,'%C','%p','%m')
```

Step#4 Run above Test class Product.java (ctrl+F11) to see output.

Example#4 SMTP Appender Example

Step#1 : Define one Test class with example code with log methods.

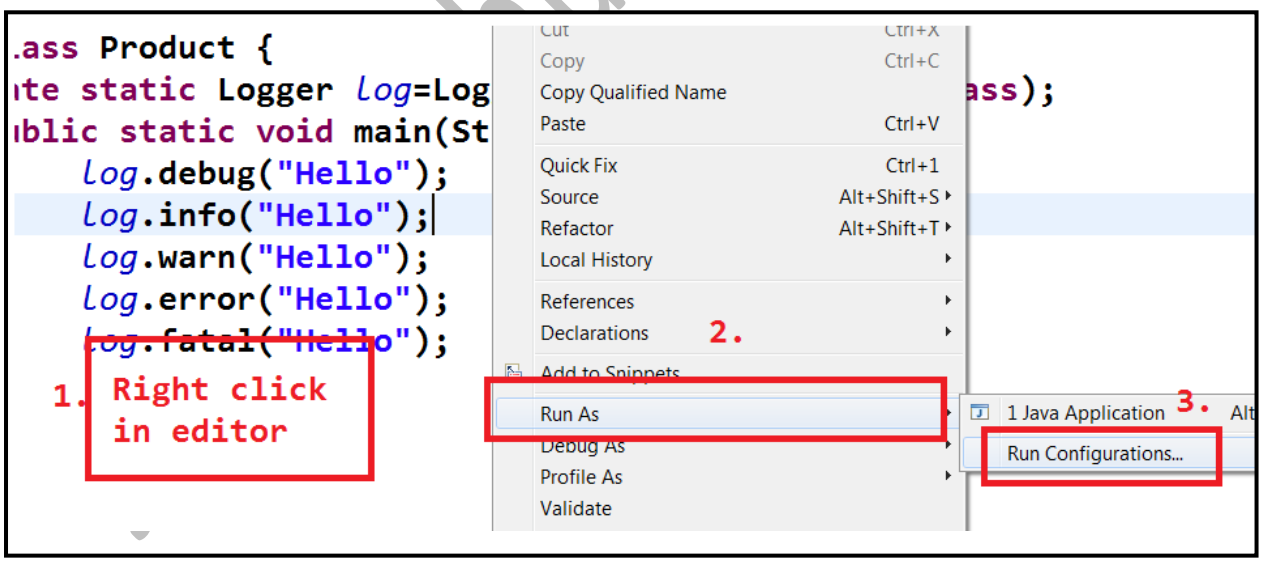
```
package com.app;
import org.apache.log4j.Logger;
public class Product {
        private static Logger log=Logger.getLogger(Product.class);
          public static void main(String[] srs){
                log.debug("Hello");
                log.info("Hello");
                log.warn("Hello");
                log.error("Hello");
                log.fatal("Hello");

        }
}
```

Step#2 Create log4j.properties file with below code

**** Modify user Name and password before you run this application**

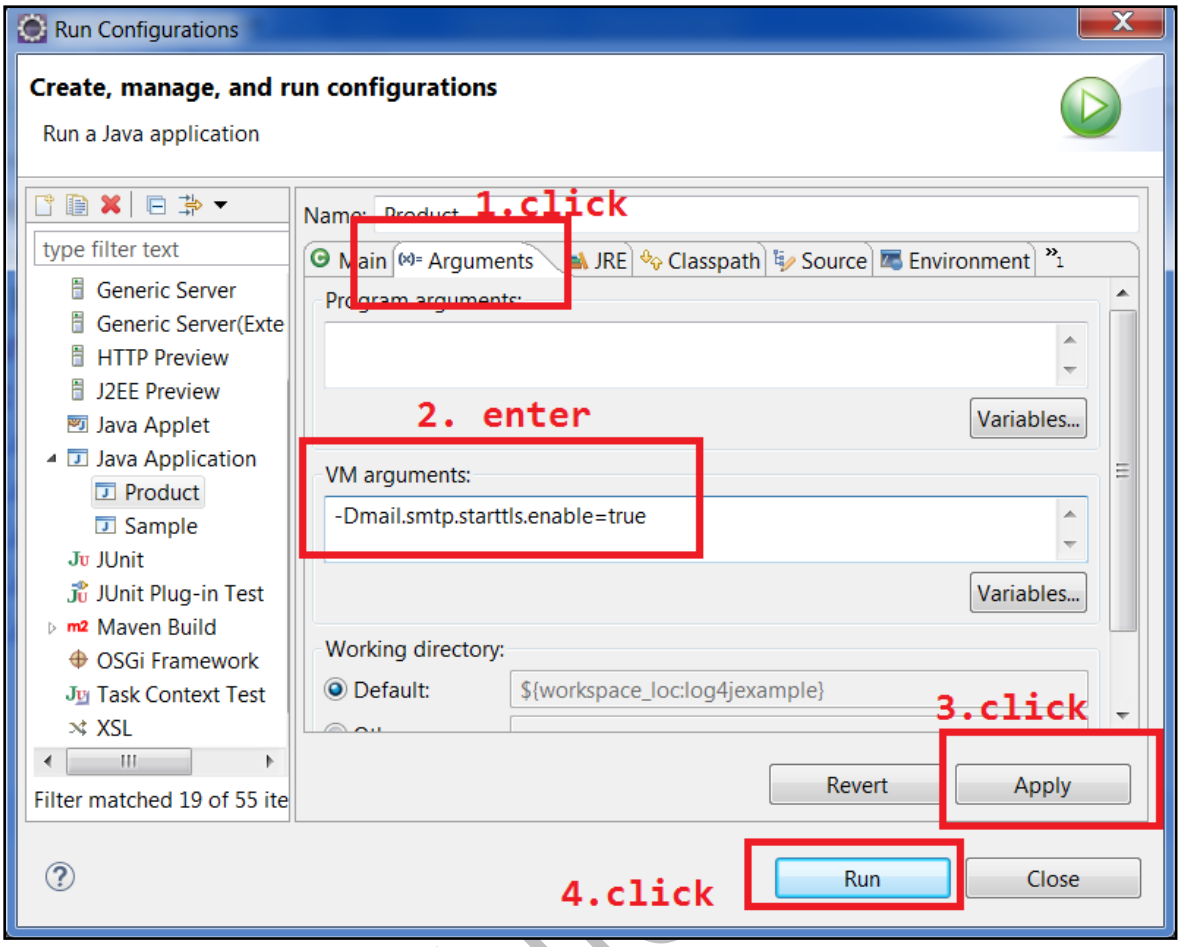
```
# Root logger option
log4j.rootLogger=DEBUG, email
#configuring the SMTP appender
log4j.appender.email=org.apache.log4j.net.SMTPAppender
log4j.appender.email.SMTPHost=smtp.gmail.com
log4j.appender.email.SMTPUsername=raghusirjava@gmail.com
log4j.appender.email.SMTPPassword=abdeyhfk@33
log4j.appender.email.From=raghusirjava@gmail.com
log4j.appender.email.To=<your email id>@gmail.com
log4j.appender.email.Subject=Log of messages
log4j.appender.email.Threshold=DEBUG
log4j.appender.email.layout=org.apache.log4j.PatternLayout
log4j.appender.email.layout.ConversionPattern= %m %n
```

Step#3 Configure this VM Argument before you run application
Right click in Editor and Choose "Run As" => "Run Configuration"



Enter this key=value in VM arguments in below screen

-Dmail.smtp.starttls.enable=true



Step#4 Run above Test class Product.java (ctrl+F11) to see output. Or Click on Run Option shown in above screen

Facebook Group: https://www.facebook.com/groups/thejavatemple

JUnit (4.X)

Coding (or) Implementation of module/task is done by developer. After Coding is done it will be submitted to Testing Team (QA=Quality Analysis Team). QA Team may do different types of testing. But before submitting code to QA, if DEV performs Individual tasks/modules tested then Error(BUG) rate will be reduced in Application.

Programmer cannot test complete project he can test only his module or task that is called as Unit Testing. Here Unit Indicates a class, method, one layer, one module etc..

Programmer can do testing in two ways

I. **Manual Approach**: This approach may not given accurate result. On testing every time it is a time consuming approach.
II. **Programming Approach**: It will give more accurate result. One time if we write code for Unit Testing, 2nd time onwards time saving approach.

Checking module, by entering input manually, writing sysouts in program and observing, de-bug for complete step by step check manually comes under Manual Testing.

JUnit is a Framework (given by apache) to test one module/task(Unit),using programming approach. It provides accurate result in testing. This JUnit always provides Test -- PASS or FAIL only.

JUnit Supports two types of Programs for Unit Testing those are:

1. **Test Case (one module/part check)** : It is a class used to write test methods which confirms the code working functionality. It returns test PASS/FAIL. On running Test case, result will be shown on JUnit Console.
2. **Test Suite (One or multiple Test cases)** : It is also a class, It is a collection of multiple Test Cases together to run all test cases at a time.

To Write JUnit Test case we need to know two important concepts. Those are Annotations in JUnit and Assert (C) API which has all test methods (static), those are also called as assert methods.

Annotations in JUnit:

a. **@Test** : It must be applied over test methods in Test Case. It will be executed by JUnit. It returns Test Pass/FAIL
b. **@Test(timeout=200)** : To avoid long time testing, or dead locks in Testing timeout option is used. After given time is over test will be considered as FAIL.
c. **@Before** : To provide basic initialization or to pre-process any logic like object creations, data setting, connections openings, Files loading etc will be done in this method. It will be executed once for every test method in Test Case (class).
d. **@After** : After Test method is executed, to execute post-process logic of a test method this annotated method is used. It will be executed once for every test method in Test Case (class).
e. **@BeforeClass** : To initialize any static data or a logic which will be executed only once will be written in this annotated method in Test Case. It will be executed only once before all test begun in Test Case. It must be applied to a static method.
f. **@AfterClass** : It will be executed after all test methods are executed in a Test Case. It is used to execute static post-process logic. That means one time post process logic of test methods. This annotated method must be static.
g. **@RunWith**(__.class) : These are used to add extra capabilities to JUnit Test classes
h. **@SuitClasses**({ .class,..}) : This annotation is used to create Test Suite.

Creating a JUnit Test case :- One test case is used to test one module or task mainly. Naming Rule is mainly followed in development bit it is optional. class name must look like "Test**<Module name>**".

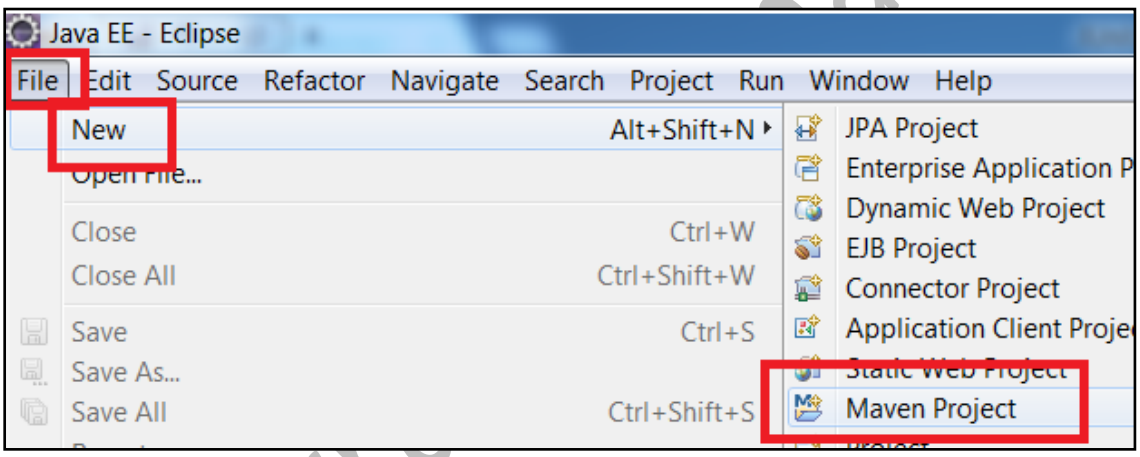
ex: for Employee module Test case is :TestEmployee. Like other examples are TestAdmin, TestLocation.

Steps to create JUnit Example Test case with eclipse screens:

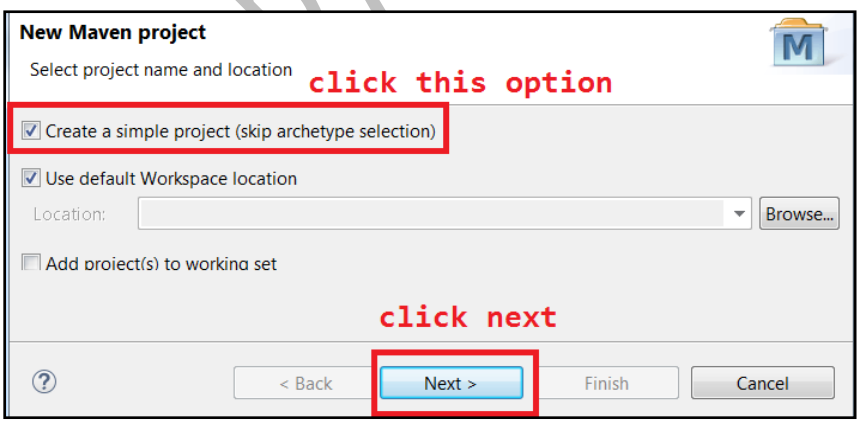
In this Example we are using a Maven project here. Maven supports in-built design of JUnit Test Programming.

a. Create one maven project & set Updated JDK/JRE.
b. Right click on "**src/test/java**" folder.
c. Choose new option and go to choose "other .."
d. Search with "JUnit" word
e. Then select "JUnit Test Case"
f. Click on next
g. Enter Name and package
h. Click on finish button.

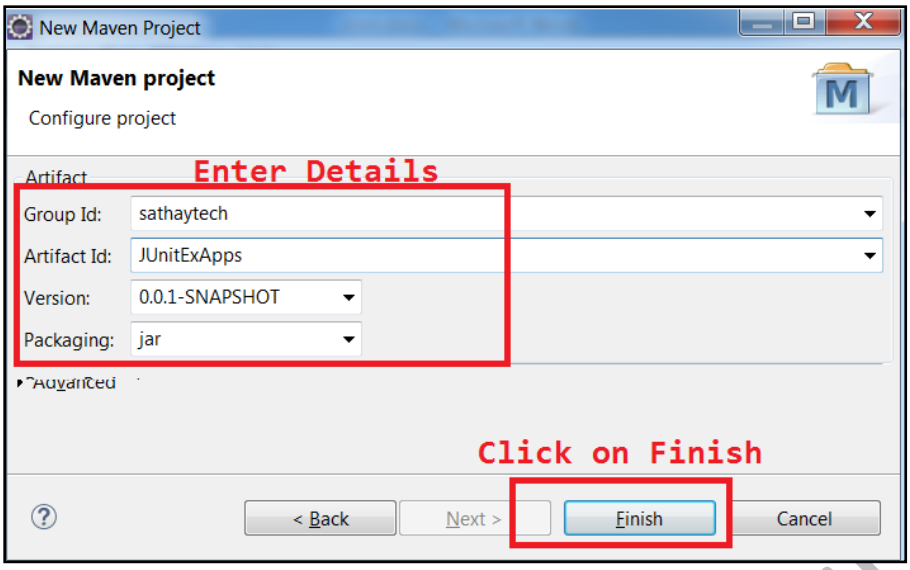
Screen : 1



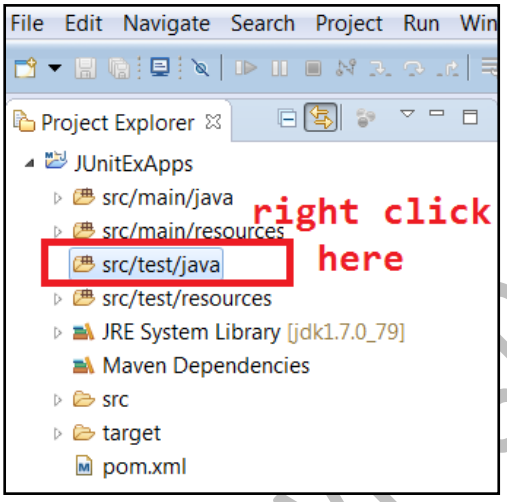
Screen : 2



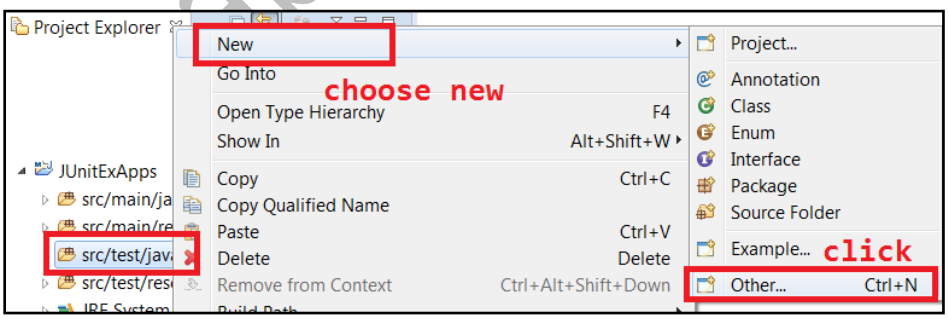
Screen : 3



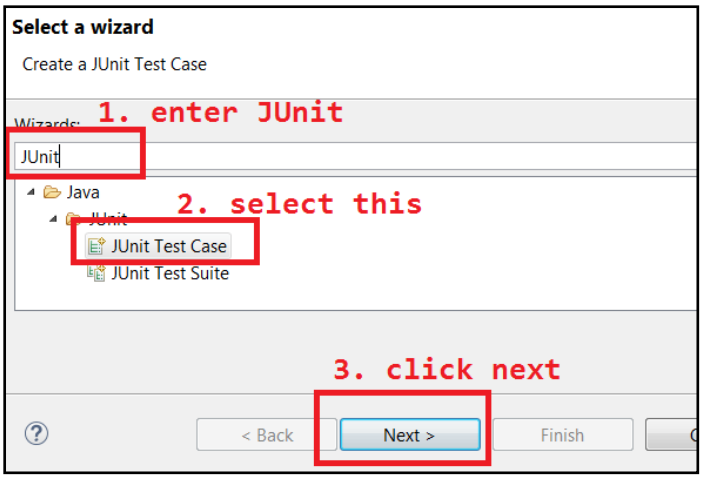
Screen : 4



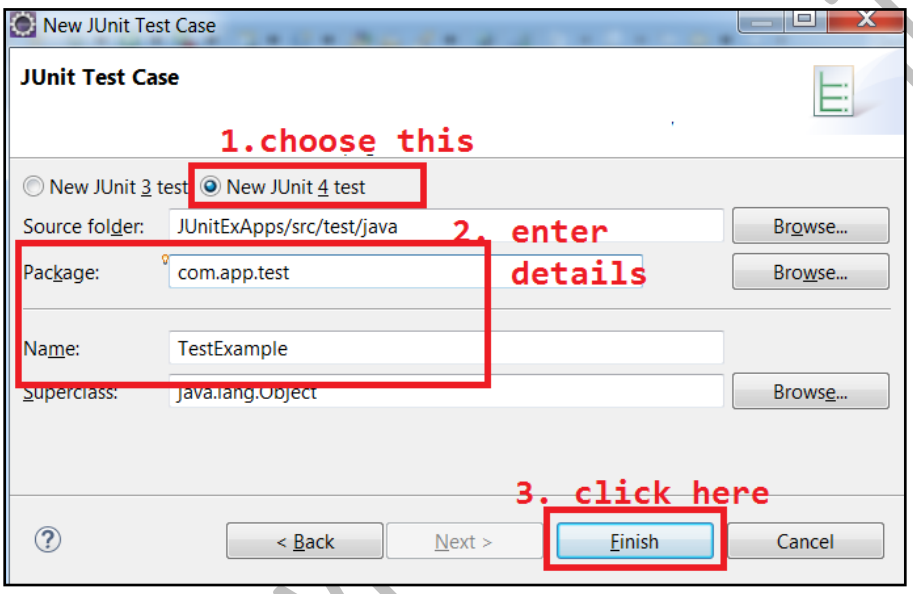
Screen : 5



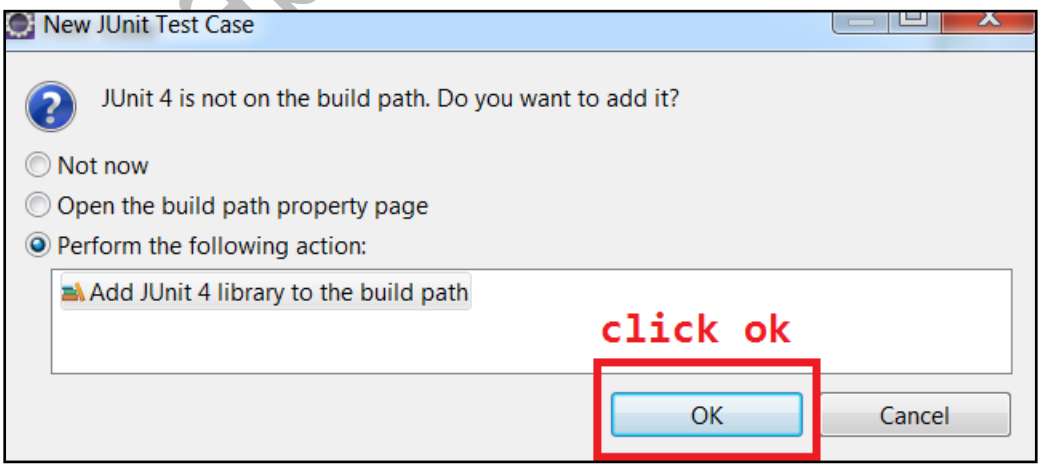
Screen : 6



Screen : 7



Screen : 8



Note:

1. Test case is a class. It contains test methods.
2. Every Test method should be public and void type. It must be annotated with @Test(org.junit package). If no annotation is provided over method then it is called as simple method.
3. We can write supportive methods for test methods like Before, After etc...
4. @Before is used to design one method which will be executed before every test method.
5. @After is used to write one method which executes after every test method.
6. To execute any logic only one time before or after all tests then use @BeforeClass and @AfterClass over methods but those methods must be static type.

Example : Test Case:-

```
package com.app.test;

import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;

public class TestExample {

	@BeforeClass
	public static void onetimeBefore(){
	 System.out.println("Before All..");
	}
	@Before
	public void preWork(){
	 System.out.println("before..");
	}
	@Test
	public void testA() {
	 System.out.println("Test-1");
	}
	@Test
	public void testB() {
	 System.out.println("Test-2");
	}
```

```
        @Test
        public void testC() {
         System.out.println("Test-3");
        }
        @After
        public void postWork(){
         System.out.println("After..");
        }
        @AfterClass
        public static void ontimeAfter(){
         System.out.println("After All..");
        }
}
```

static import in java: To use one class in another class, if both are in different packages, then we must write import statement. import syntax:

import package.className;

If we use import statement , it will import all members (static and non-static) Sometimes we need only static data then if we use normal import it will load all members and waste then memory hence app performance will be reduced. To load only static members (static variables and static methods) use static import concept.

static import syntax:

```
import static  pack.class.*;
--or--
import static  pack.class.member;
```

Here member means variable or method.

Example:

```
package com.app;
public class A{
  int id;
  String name;
  static int pid =3;
```

```
  void  m1() { ... }
  static void m2(){ ... }
}
```

From above class if we want load only static propeties into JVM then use
" import static com.app.A.* "

```
package com.one;
import static com.app.A.*;
public class B{
 void show() {
  sysout(pid);
  m2(); //method call
 }
}
```

Assert (org.junit) JUnit API

To validate any application Logic we use Assert API methods, which are also called as UnitTesting methods.

All these methods are static and void methods. These throws Assertion Error in fail These methods will tell "Is Test PASS or FAIL?" only.

After executing logic it will return some output(actual result) it can compared with expected result. If valid then PASS else FAIL, in this way JUnit Testing will be done

Every Assert method is overloaded to define user friendly Error Messages. All assert methods are given as,

- assertEquals("exp", "original");
- assertEquals("MSG","exp", "org");
- assertNotEquals("exp", "org");
- assertNotEquals("MSG","exp", "org");
- assertTrue(5>(8+9)/5)
- (boolean condition)
- assertTrue("Seems False..",5>(8+9)/5);

- assertFalse(5>(8+9)/5);
- assertFalse("Might be True",5>(8+9)/5);
- fail();fail("Falied method test");
- assertSame(ob1, ob2);
- assertSame("not same..",ob1, ob2);
- assertNotSame(ob1, ob2);
- assertNotSame("same..",ob1, ob2);

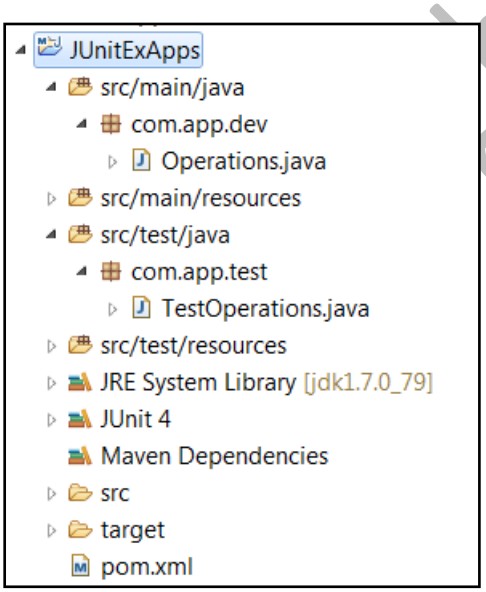
Here assert=Expected [(or) Expected Result Is]. assertSame checks hashCodes, not data. assertEquals checks data.

Example#1:

Requirement : Develop one class with method that performs sum operation.

Test : Test Above method using JUnitTest case with some random data.

Folder Structure:



Code : Requirement

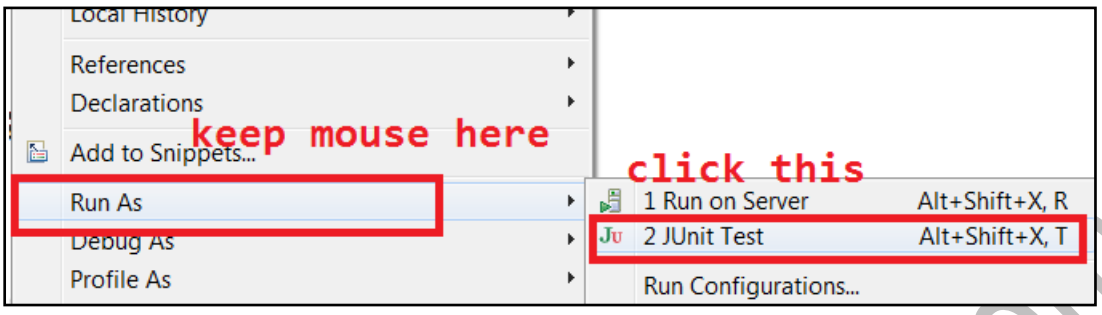
```
package com.app.dev;

public class Operations {
```

```
	public int doSum(int x,int y){
		return x+y;
	}
}
```

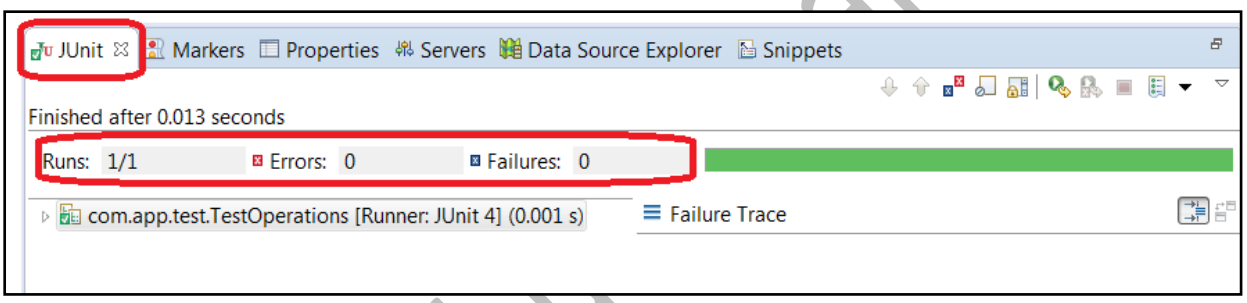
Create one JUnit Test case to verify that above code written properly or not.

```
package com.app.test;

import static org.junit.Assert.assertEquals;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;

import com.app.dev.Operations;

public class TestOperations {
	Operations opr=null;
	int val1,val2,exp;

	@Before
	public void preSetup(){
		opr=new Operations();
		val1=val2=1;
		exp=2;
	}

	@Test
	public void test() {
		int res=opr.doSum(val1, val2);
		assertEquals(
				"Dosum is not working",exp,res);
	}

	@After
	public void postTest(){
		opr=null; val1=val2=exp=0;
	}
}
```

Run As JUnit Test: Right click on Editor then



* Result will be printed on JUnit Console: Observe no.of Test PASS/FAIL.



Method Execution Order:

JUnit By Default takes care of executing methods in its own order (DEFAULT) it can be deterministic, but not predictable(no 100% confirmation) JUnit also supports JVM Method Sorting (MethodSorters.JVM) or we can go for method name sorting as (MethodSorters.NAME_ASCENDING)

Example:-

```
@FixMethodOrder(MethodSorters.NAME_ASCENDING)

public class TestSample {

  ....// @Test

}
```

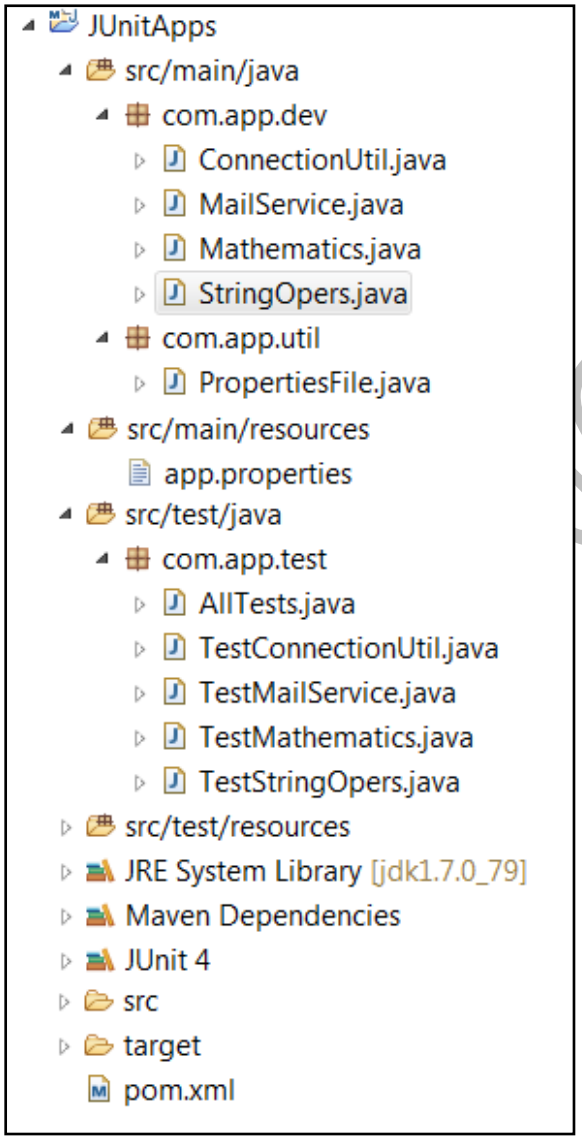
Example JUnit Application with Requirement and JUnit Test cases

Requirements : Write An **Application** and Test Case using JUnit for

1. Singleton Database connection check
2. Sending Email using Java Mail API
3. Find input Number is Armstrong number or not.
4. Input String following ABC Cycle or not

Application code followed by JUnit Test case code:-

Folder structure:



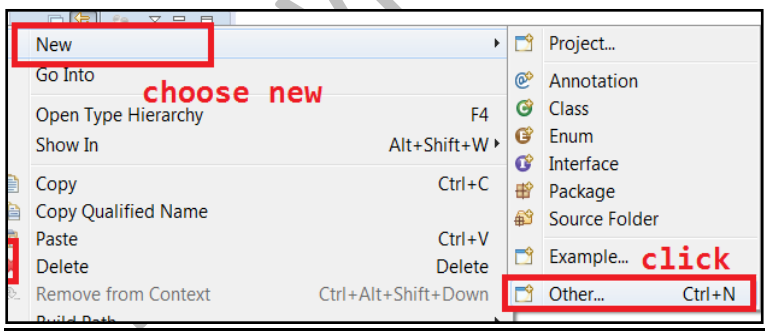
pom.xml code:-

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

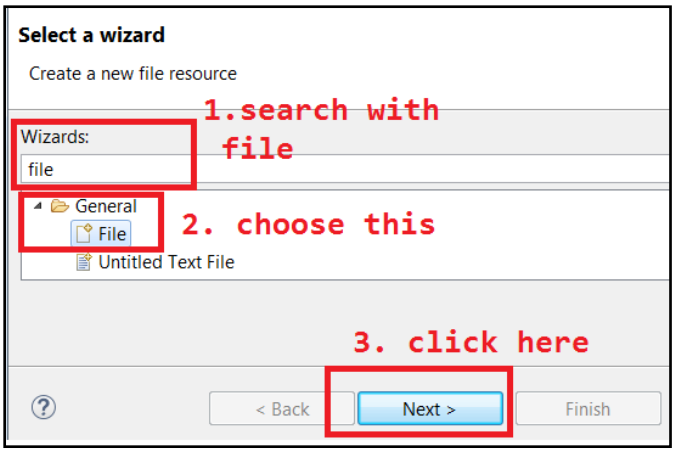
```
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <groupId>sathatech</groupId>
        <artifactId>JUnitApps</artifactId>
        <version>1.0</version>

        <dependencies>
                <dependency>
                        <groupId>mysql</groupId>
                        <artifactId>mysql-connector-java</artifactId>
                        <version>5.1.6</version>
                </dependency>

                <dependency>
                        <groupId>javax.mail</groupId>
                        <artifactId>mail</artifactId>
                        <version>1.4</version>
                </dependency>

        </dependencies>
</project>
```

Create one Properties for all key=value, for email and Database connection.

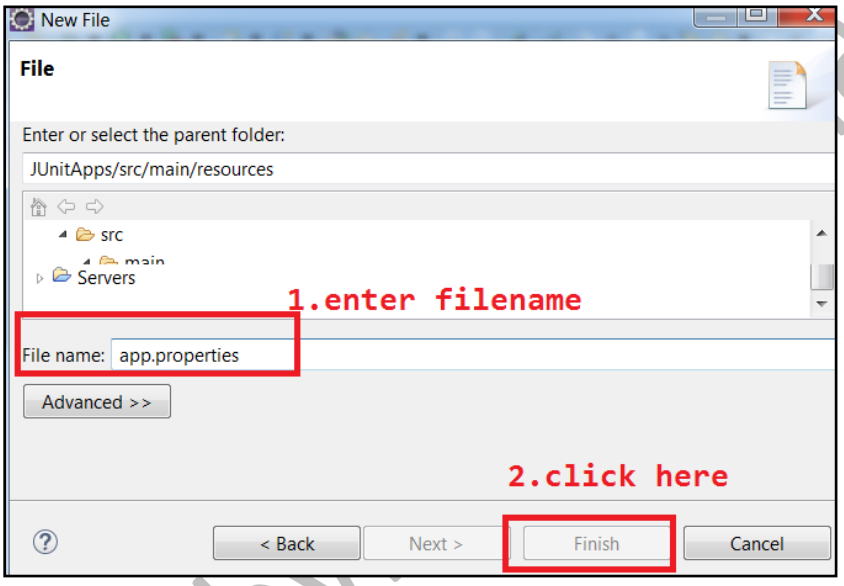
Right click on "src/main/resources" folder , then goto new and select other



Search with "file" option and select file , click next button



Enter File name ex: app.properties and click finish button



app.properties (code)

```
#DB Properties
dc=com.mysql.jdbc.Driver
url=jdbc:mysql://localhost:3306/test
un=root
pwd=root

#mail
username=raghusirjava@gmail.com
password=Asdfghjkl1@
AuthKey=mail.smtp.auth
TLSKey=mail.smtp.starttls.enable
HostKey=mail.smtp.host
```

```
PortKey=mail.smtp.port
```

1. Code for : Singleton Database connection check

```
package com.app.dev;

import static com.app.util.PropertiesFile.*;

import java.sql.Connection;
import java.sql.DriverManager;


public class ConnectionUtil {

	private static Connection con=null;
	static{
		try {
			Class.forName(getProperties().getProperty("dc"));

	con=DriverManager.getConnection(getProperties().getProperty("url"),
					getProperties().getProperty("un"),
getProperties().getProperty("pwd"));
		} catch (Exception e) {
			e.printStackTrace();
		}

	}
	public static Connection getSingeltonConnection(){
	    return con;
	}
}
```

JUnitTest Case code:

```
package com.app.test;


import static org.junit.Assert.*;
import java.sql.Connection;

import org.junit.After;
```

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

import com.app.dev.ConnectionUtil;

public class TestConnectionUtil {

	Connection con1,con2;

	@Before
	public void setUp(){

		con1=ConnectionUtil.getSingeltonConnection();
		con2=ConnectionUtil.getSingeltonConnection();
	}

	@Test
	public void test() {
		assertNotNull(con1);
		assertNotNull(con2);
		assertSame("Not a singleton connection",con1, con2);
	}
	@After
	public void clear(){
		con1=con2=null;
	}
}
```

2. Sending Email using Java Mail API:

```
package com.app.dev;

import static com.app.util.PropertiesFile.*;
import java.util.Properties;

import javax.mail.Message;
import javax.mail.MessagingException;
import javax.mail.PasswordAuthentication;
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.internet.InternetAddress;
```

```
import javax.mail.internet.MimeMessage;

public class MailService {

	public static boolean sendEmail(String toAddr,String subject,String text)
{
		boolean isMailSent=true;
		final String username = getProperties().getProperty("username");
		final String password = getProperties().getProperty("password");

Properties props = new Properties();
props.put(getProperties().getProperty("AuthKey"), "true");
props.put(getProperties().getProperty("TLSKey"), "true");
props.put(getProperties().getProperty("HostKey"), "smtp.gmail.com");
props.put(getProperties().getProperty("PortKey"), "587");

Session session = Session.getInstance(props,
	new javax.mail.Authenticator() {
	protected PasswordAuthentication
	 getPasswordAuthentication() {
	return new PasswordAuthentication(username, password);
			}
		});

try {

 Message message = new MimeMessage(session);
 message.setFrom(new InternetAddress(
				getProperties().getProperty("username")));
message.setRecipients(Message.RecipientType.TO,InternetAddress.parse(toAd
dr));
message.setSubject(subject);//subject
message.setText(text);//message

Transport.send(message);
System.out.println("Done");

} catch (MessagingException e) {
			isMailSent=false;
			e.printStackTrace();
```

```
		}
		return isMailSent;
	}
}
```

JUnit Test case Code:-

```
package com.app.test;

import static org.junit.Assert.*;

import org.junit.After;
import org.junit.Before;
import org.junit.Test;

import com.app.dev.MailService;

public class TestMailService {

	String toAddr,sub,text;

	@Before
	public void setUp(){
		toAddr="javabyraghu@gmail.com";
		sub="TESTAAA";
		text="HEllo..";
	}
	@Test(timeout=5*1000)
	public void test() {
	boolean flag=MailService.sendEmail(toAddr, sub, text);
		assertTrue(flag);
	}
	@After
	public void clean(){
		toAddr="javabyraghu@gmail.com";
		sub="TEST";
		text="HEllo..";
	}
}
```

3. Find input Number is Armstrong number or not:

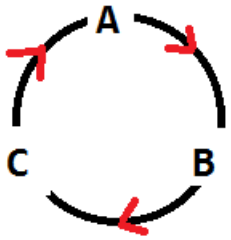
```
package com.app.dev;

public class Mathematics {

	public static boolean isArmStrong(int num){
		int temp=num,c;
		int armNum=0;
		int len=Integer.toString(num).length();
		while(num>0){
			c=num%10;
			armNum= armNum+ (int)Math.pow(c, len);
			num=num/10;
		}
		return temp==armNum;
	}

}
```

JUnit Test Case code:

```
package com.app.test;

import static org.junit.Assert.*;

import org.junit.After;
import org.junit.Before;
import org.junit.Test;

import com.app.dev.Mathematics;

public class TestMathematics {

	int num;
	@Before
	public void preSet(){
		num=548834;
	//http://mathworld.wolfram.com/NarcissisticNumber.html
	}
```

```
	@Test
	public void test() {
		boolean flag=Mathematics.isArmStrong(num);
		assertTrue(flag);
	}
	@After
	public void postModify(){
		num=0;
	}

}
```

4. Input String following ABC Cycle or not: String should follow Cycle




```
package com.app.dev;

public class StringOpers {

	public static boolean isFollwingABCCycle(String str){
		char[] arr=str.toCharArray();
		boolean flag=true;
		for (int i=0;i<arr.length-1;i++) {
			if(!(getNext(arr[i]).equals(arr[i+1]))) {
				flag=false;
				break;
			}
		}

		return flag;
	}
	private static Character getNext(char c){
		if(c=='A') c= 'B';
		else if(c=='B') c= 'C';
		else if(c=='C') c= 'A';
```

```
            return c;
        }

}
```

JUnit Test Case Code:-

```
package com.app.test;

import static org.junit.Assert.assertTrue;

import org.junit.After;
import org.junit.Before;
import org.junit.Test;

import com.app.dev.StringOpers;

public class TestStringOpers {

        String str;
        @Before
        public void setStr(){
                str="CABCA";
        }
        @Test
        public void test() {
                assertTrue(StringOpers.isFollwingABCCycle(str));
        }
        @After
        public void clean(){
                str=null;
        }

}
```


Test Suite :

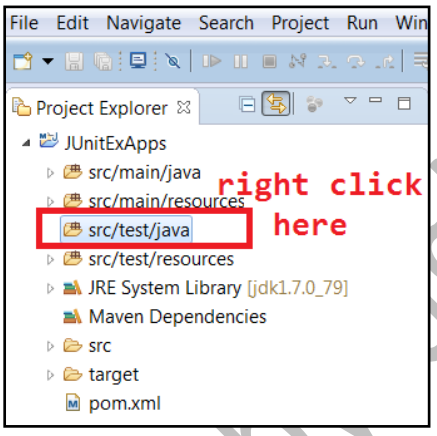
In JUnit, only one Test cases can be run at a time. If we want to run multiple Test cases at a time , then we need to create a JUnit Test suite. Every Suite must contain minimum one Test Case to Run.

Suite Also Follows naming convention which also optional. Test suite also a class but it has no logical method to execute. Test Suite class name must be suffixed with Tests word. Example All**Tests**.

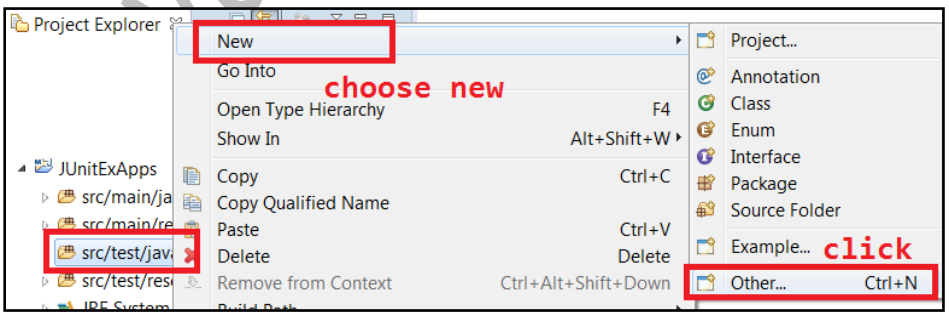
Creating JUnit Test Suite:

a. Right click on "src/test/java"
b. Choose new option
c. Click on other
d. Search using JUnit Test
e. Select JUnit Test Suite
f. Enter package and name
g. Select JUnit Test Cases to be in Test Suite
h. Click on finish

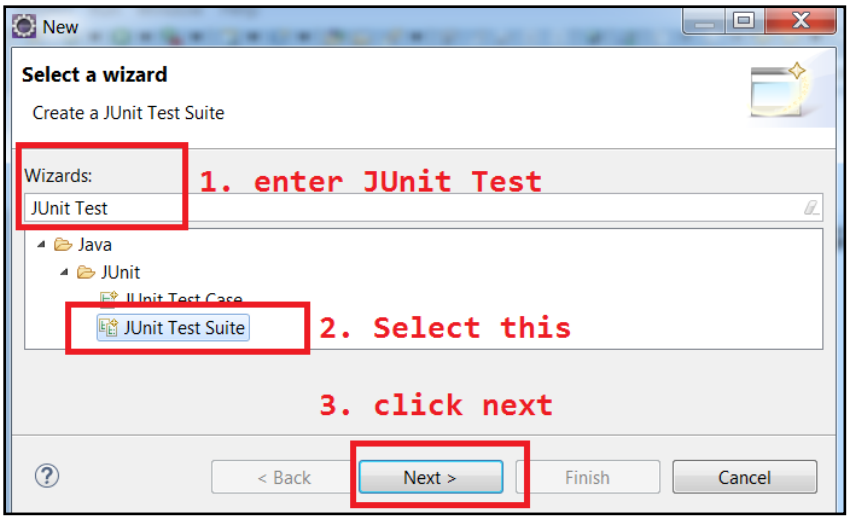
Screen : 1



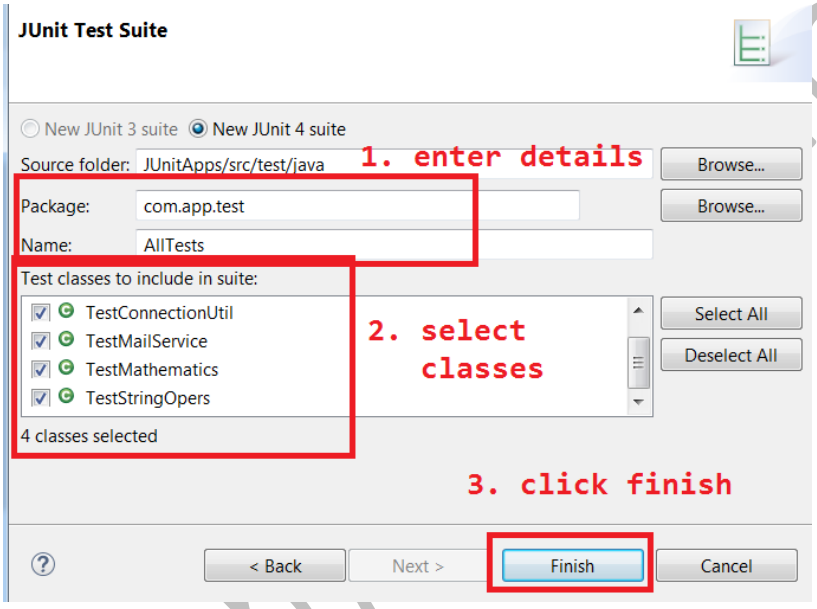
Screen : 2



Screen : 3



Screen : 4

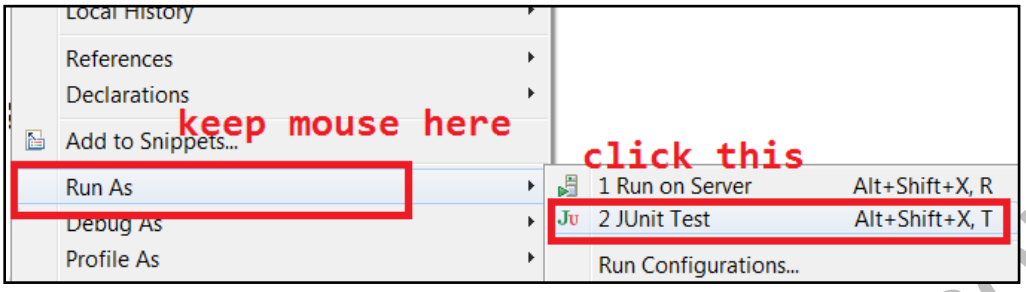


It can written with two simple annotations. Example shown below.

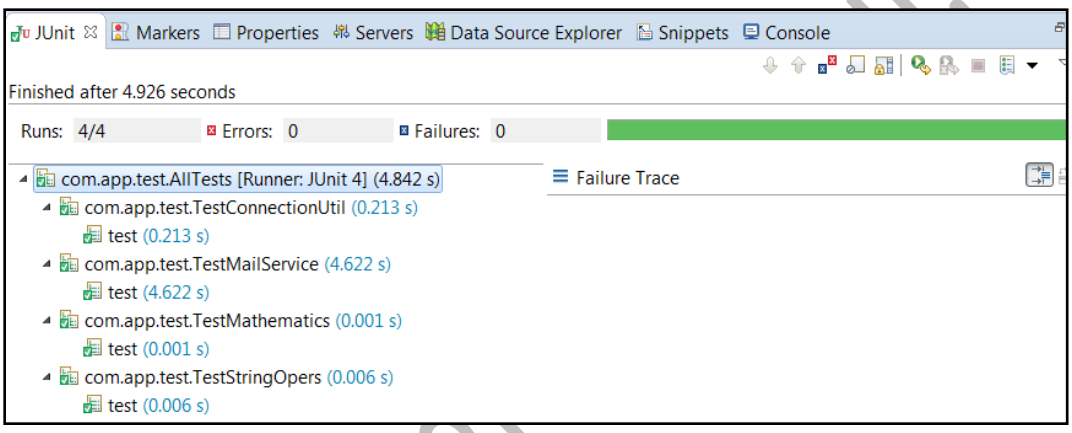
```
package com.app.test;

import org.junit.runner.RunWith;
import org.junit.runners.Suite;
import org.junit.runners.Suite.SuiteClasses;

@RunWith(Suite.class)
@SuiteClasses({ TestConnectionUtil.class, TestMailService.class,
TestMathematics.class, TestStringOpers.class })
public class AllTests {
```

```
}
```

To Run Suite : Right click on code (Editor) , choose Run As, JUnit Test.



output on JUnit Console:



GIT - CODE REPOSITORY TOOL (VERSION MANAGEMENT)

To develop app. we write different types of files in projects (like java, xml, jsp etc..). All these files are stored finally in a central server i.e. Code Repository. It maintains the details of code. like

- ✓ How many times code is modified?
- ✓ Who modified what code(+/- added/removed)
- ✓ Changes done on what date and time? Maintaining these details is known as Version Management/Version Control.

a. In developement mostly used tool is GIT(gitHub).
b. It maintains two level repository process for cross verify and commit.
c. For this we need to create an account in github.com(using email id and verify email after creation).
d. It maintains staging area to select /verify/comment the file while adding to local repository.

Git Process:

Git Repository Types:

a. Public : Free for every one
b. Private : Paid version.

Remote Repository(Steps):

a. goto https://github.com/
b. SignUp(Register) with details
c. Goto Email -and verfiy Link
d. login with un and pwd
e. Create Repository Type Public (Companies uses- Private Type)
f. Copy Git Link: https://github.com/abcd/testVen.git it is secured with un,pwd.

Eclipse Workspace and Local Repository Steps:

In eclipse,

1. Go to window menu

2. Show Views search for "GIT" select GitRepository and GitStaging.
3. Click on Window
4. showView
5. History.

Creating local repository:

1. Right Click on Project
2. choose "Team" option
3. Share project
4. select checkbox "create or use repository"
5. select option shown below
6. click create repository button
7. finish

GIT Operations with Flow:

- add file to git staging (Click-drag-drop).
- Commit and Push
- Paste Git Link in URI input box
- enter userName and password.
- next/next/finish.
- right click on project
- Choose team
- Select pull, then ,rebase to update local from remote.

Some Interview Questions over GIT:

1) Can you provide GIT/Client/Apps URLs? No, Sorry.all details what you are asking is private and confidential. I cannot provide those.

2)Who created your git account ?

Git administrator name-"XYZ", in my company. He handles all permission and auth. details.

3)What type of Account Repository areyou using?

It is private account handled by company.

4)When you check-in or push your code?

After Implementation of complete module and UnitTesting code in my local, I'll push to remote.

5)How will you find about one file all modification??

We use history option in eclipse

->right click->team->shown in history.

we can also choose any two files to compare with each other ex:select

any history options->right click->compare with each other.

6)How do you identify code modifications in GIT?

Using Symbols (+) code is added (-) code is removed.

7)What is the difference between commit and push?

commit: update code from workspace to Local Repository

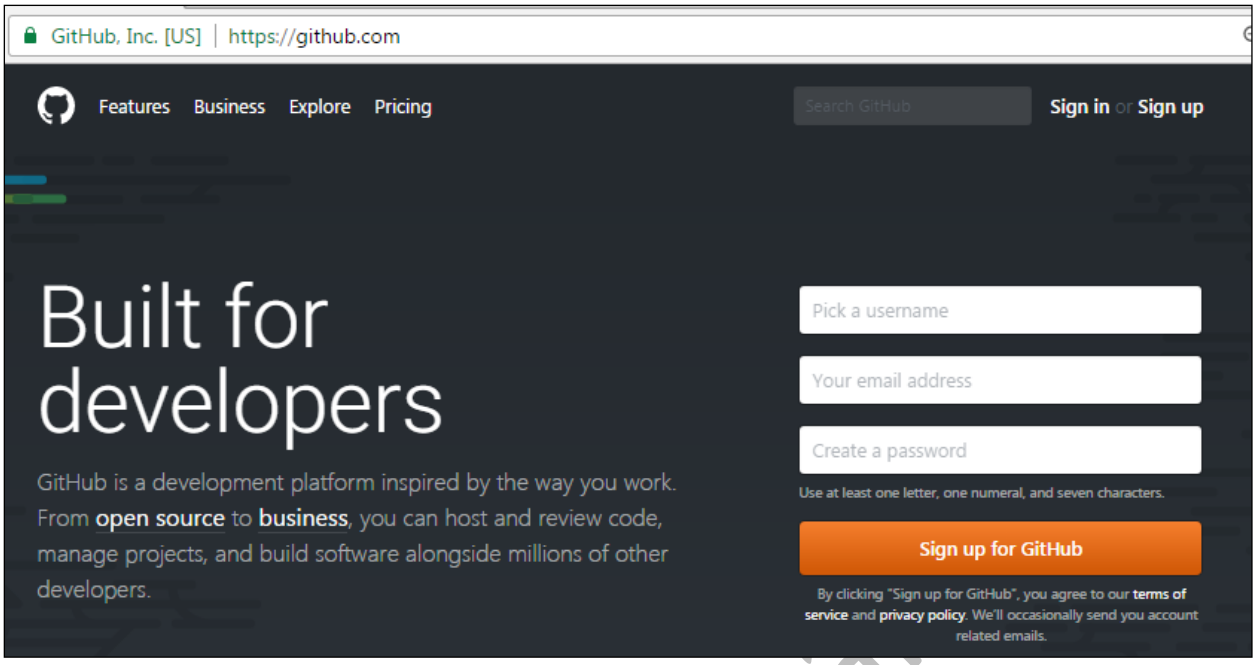
push :Update code from local to remote repository.

8)How can you update you workspace/local with remote?

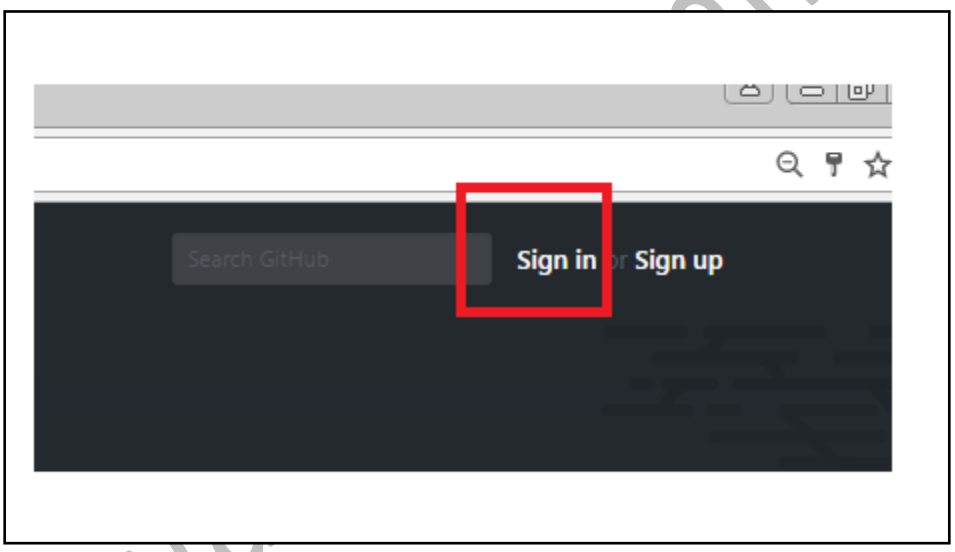
By using pull and rebase operations.

Screens Help For Complete Process:

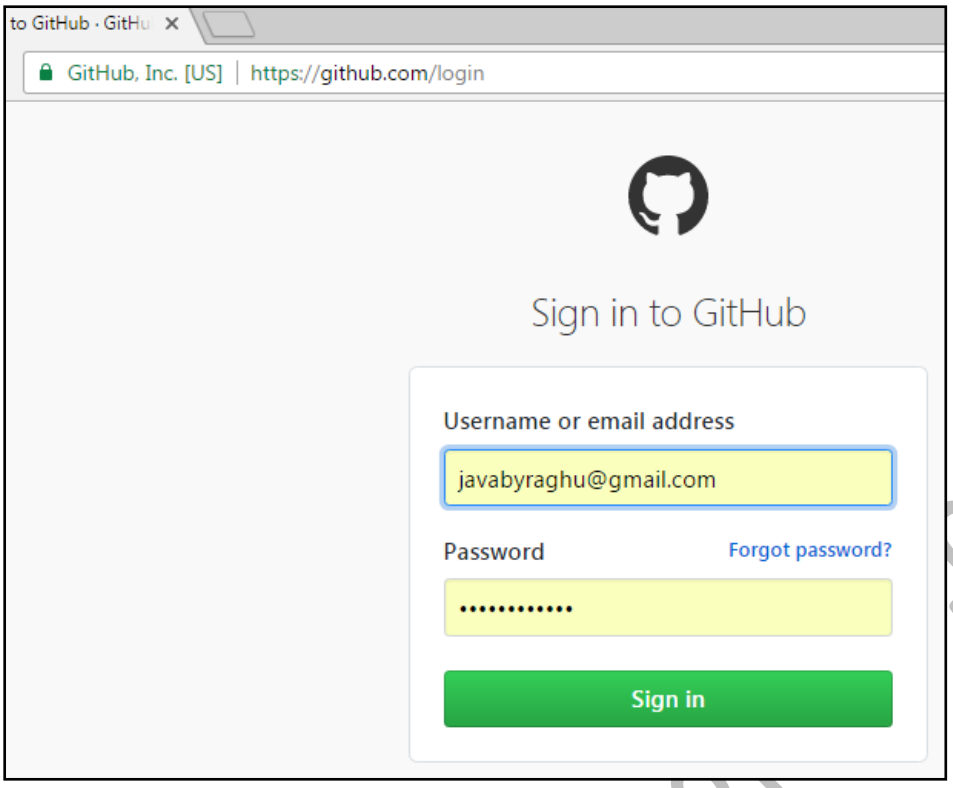
https://github.com/



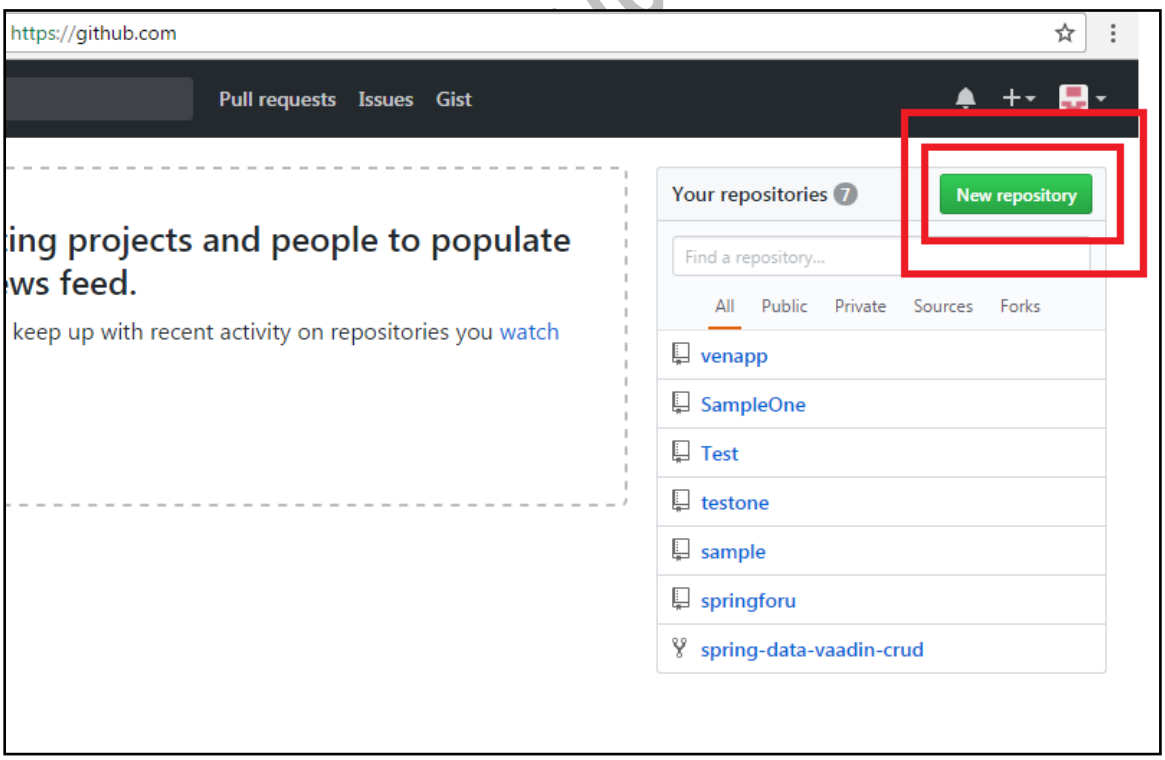
Goto email account and click on verify link. Click on signin



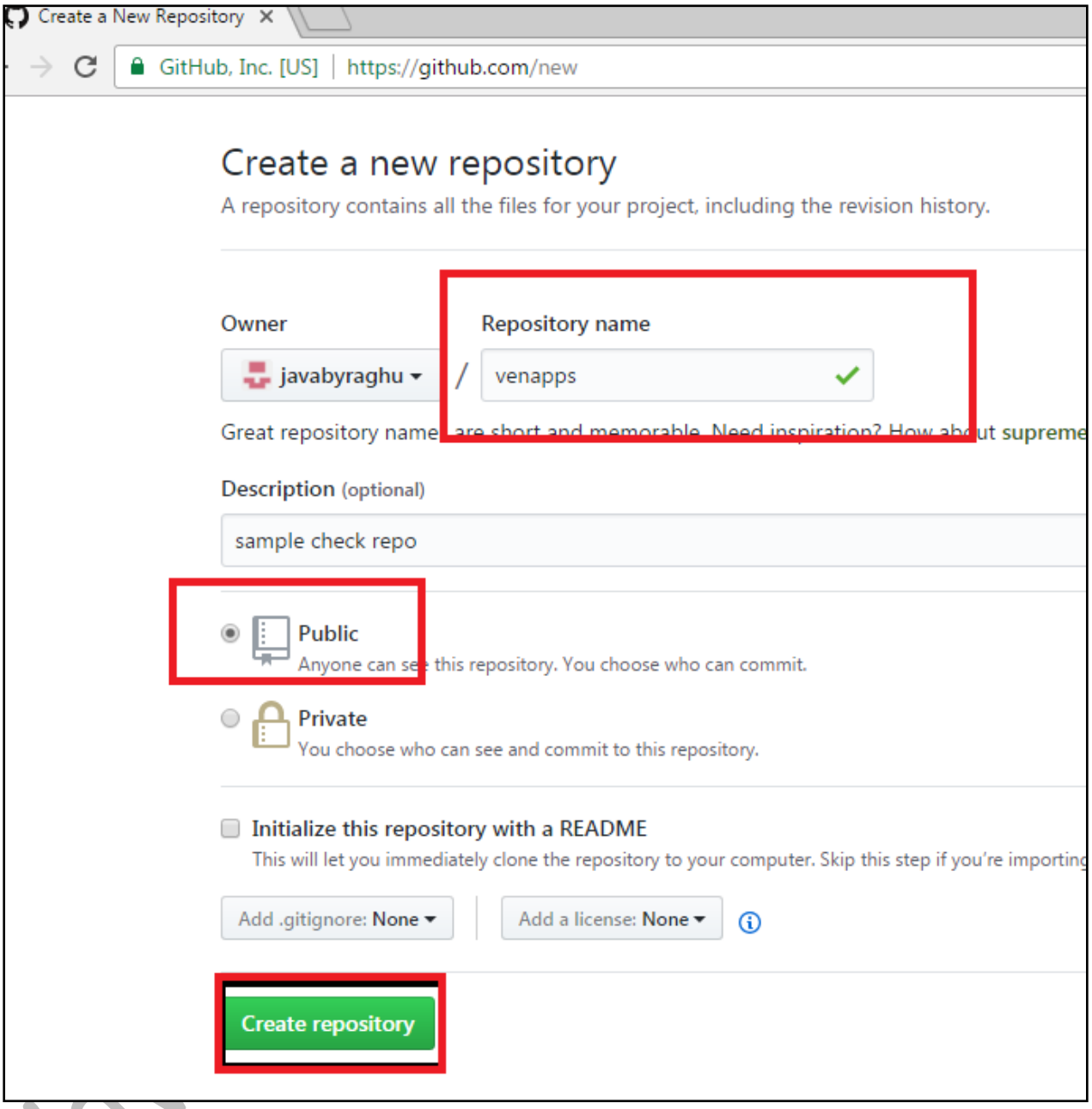
Enter user name and password.



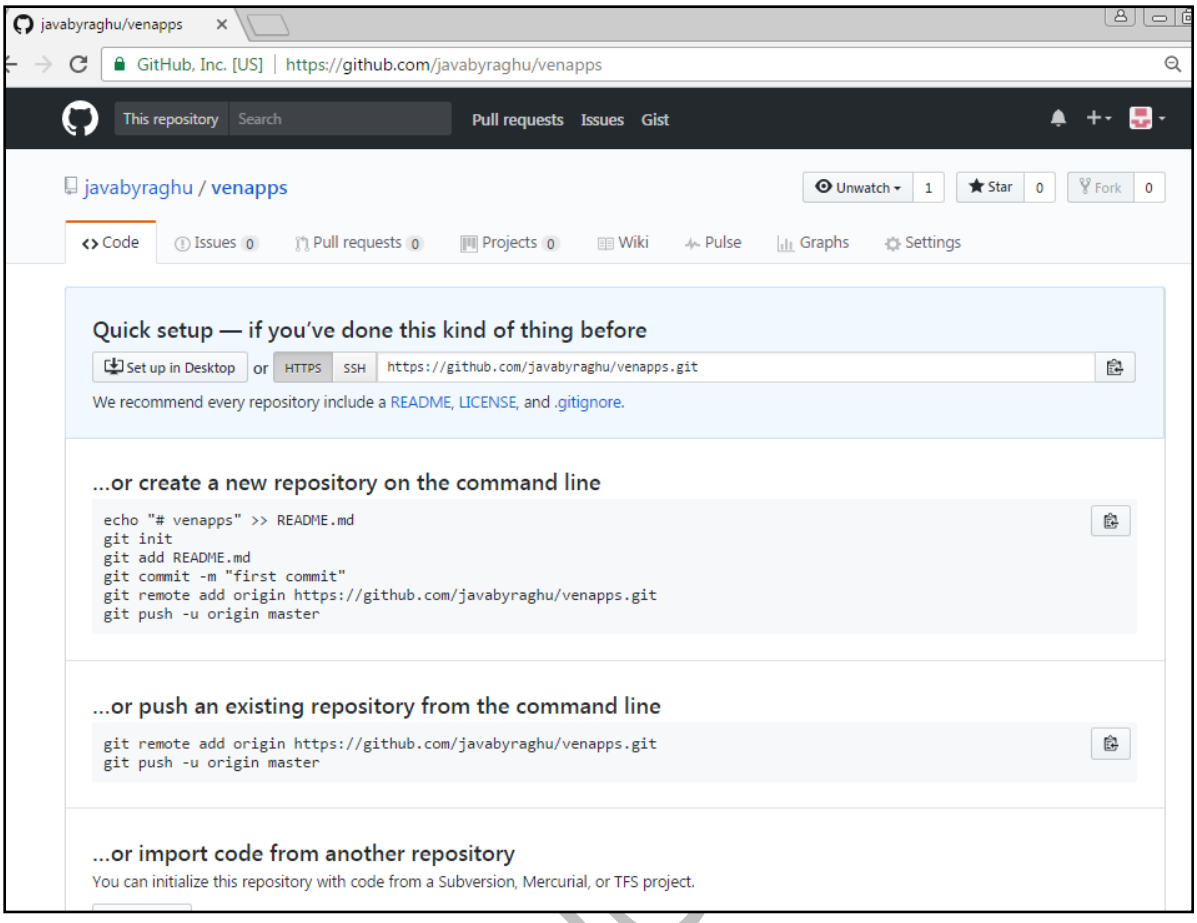
Click on new Repository

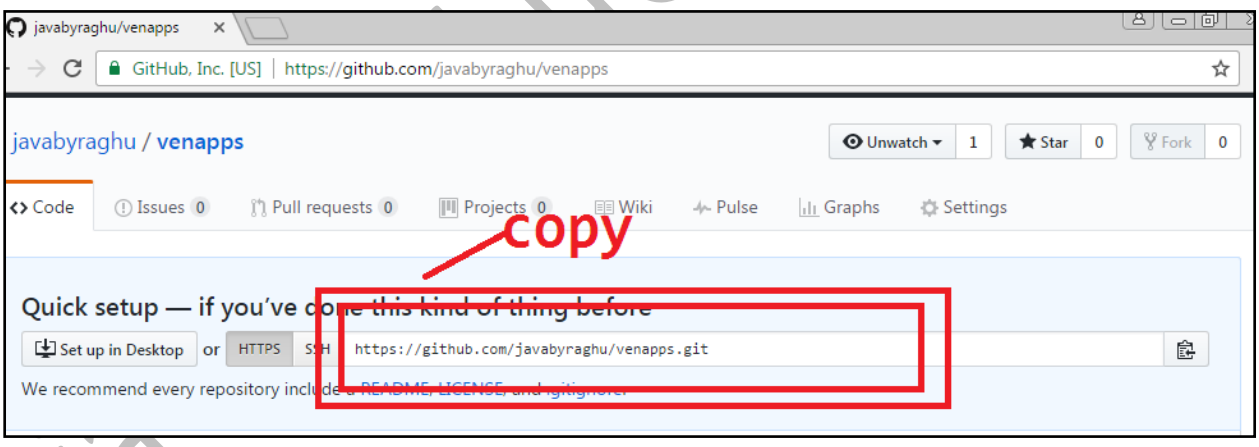


Provide Repository Details



On click create repository



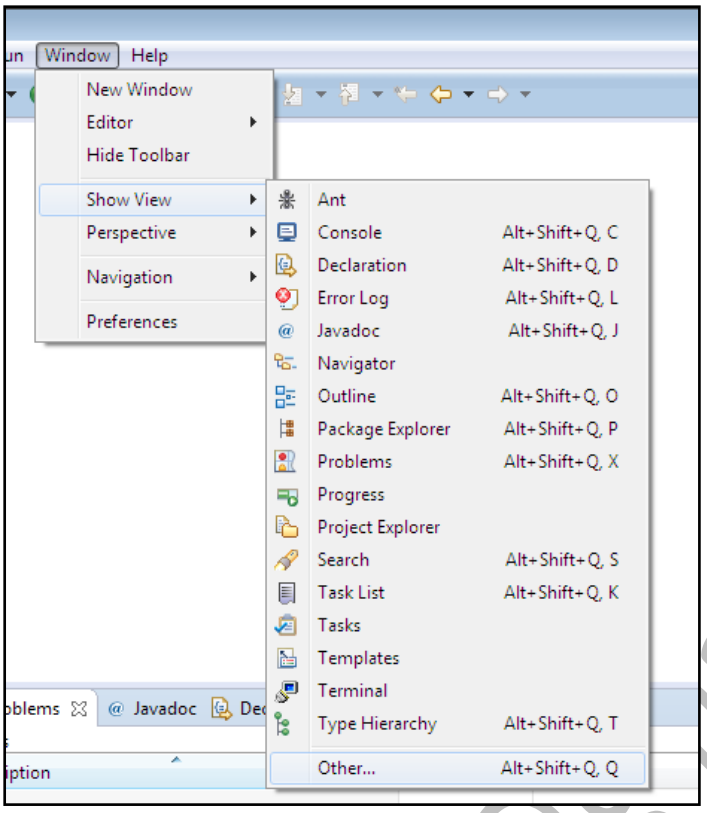


Ex : (Repository Link)

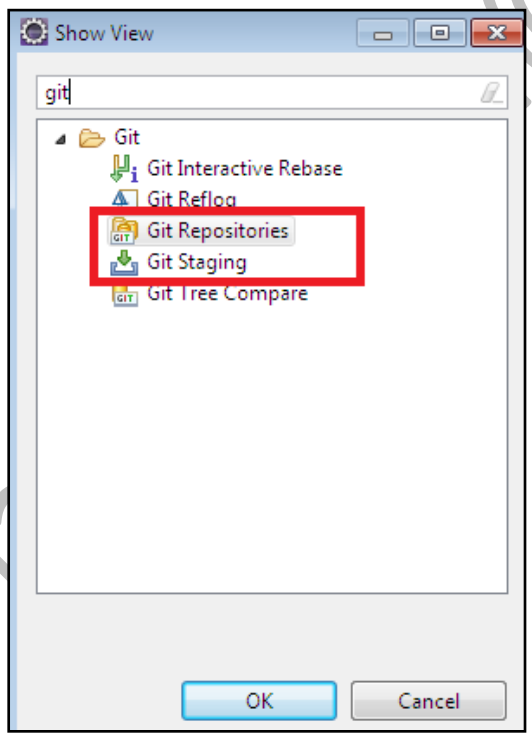
https://github.com/javabyraghu/venapps.git

Got Eclipse:

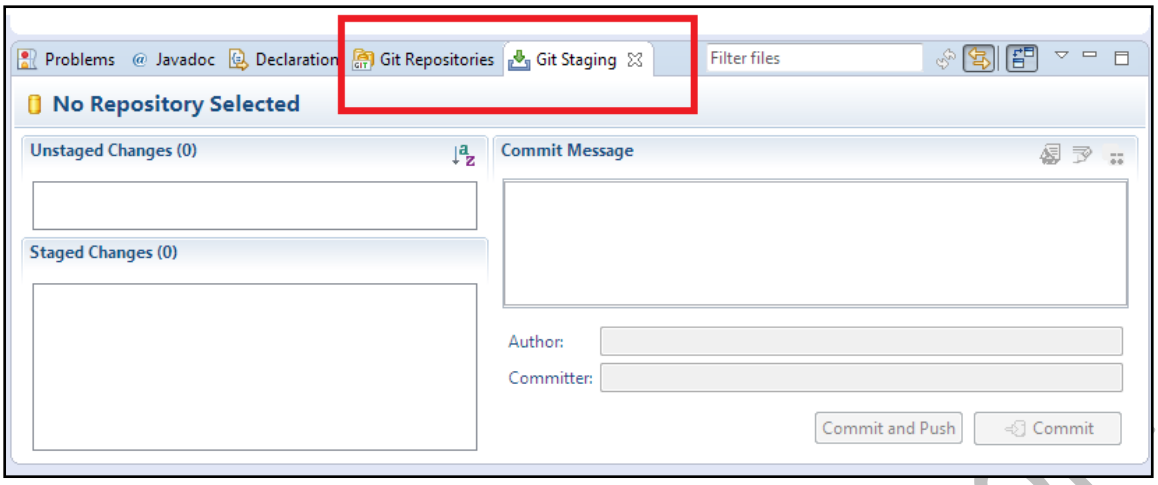
- Create A new Java Project **(project name and repository name Should be same)**
- Add Git Views



Serach with Git, select Git Repository, Staging

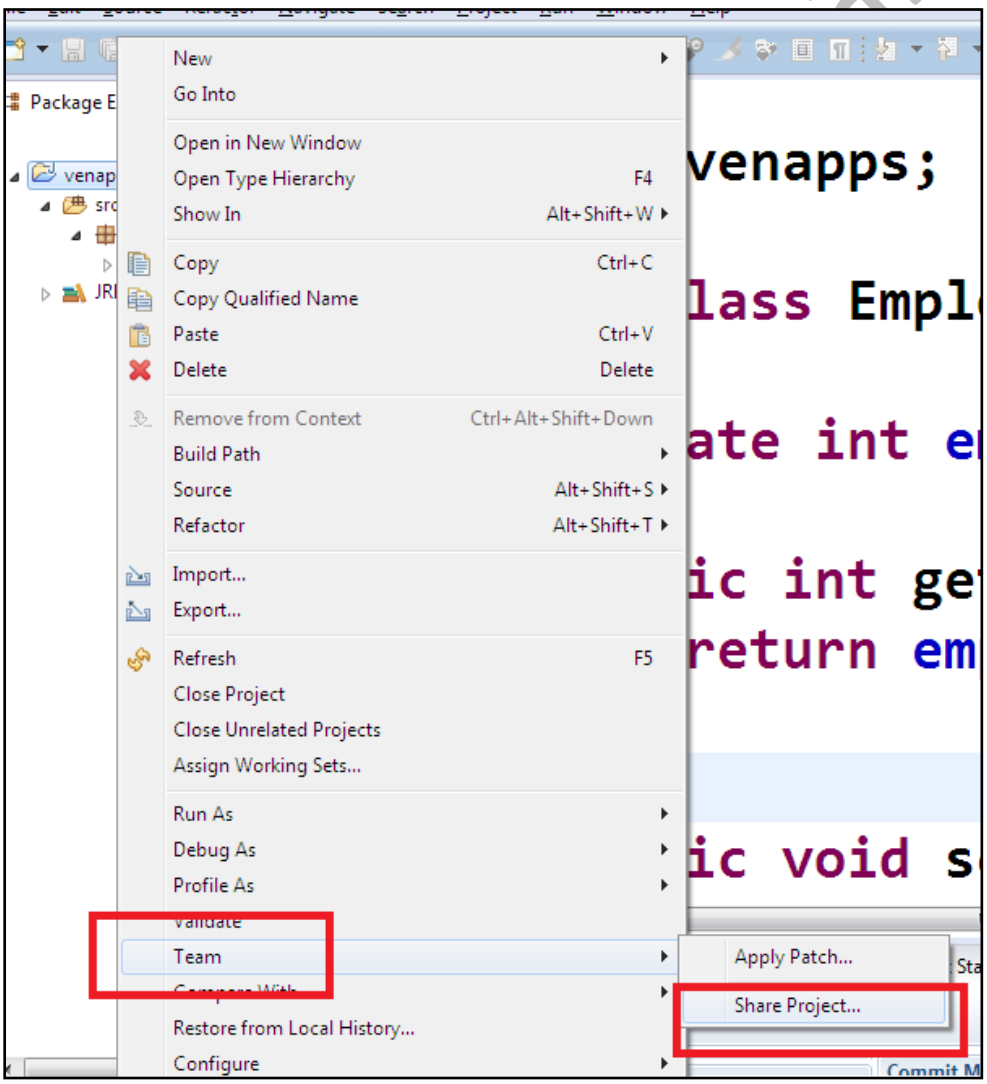


Observer options like

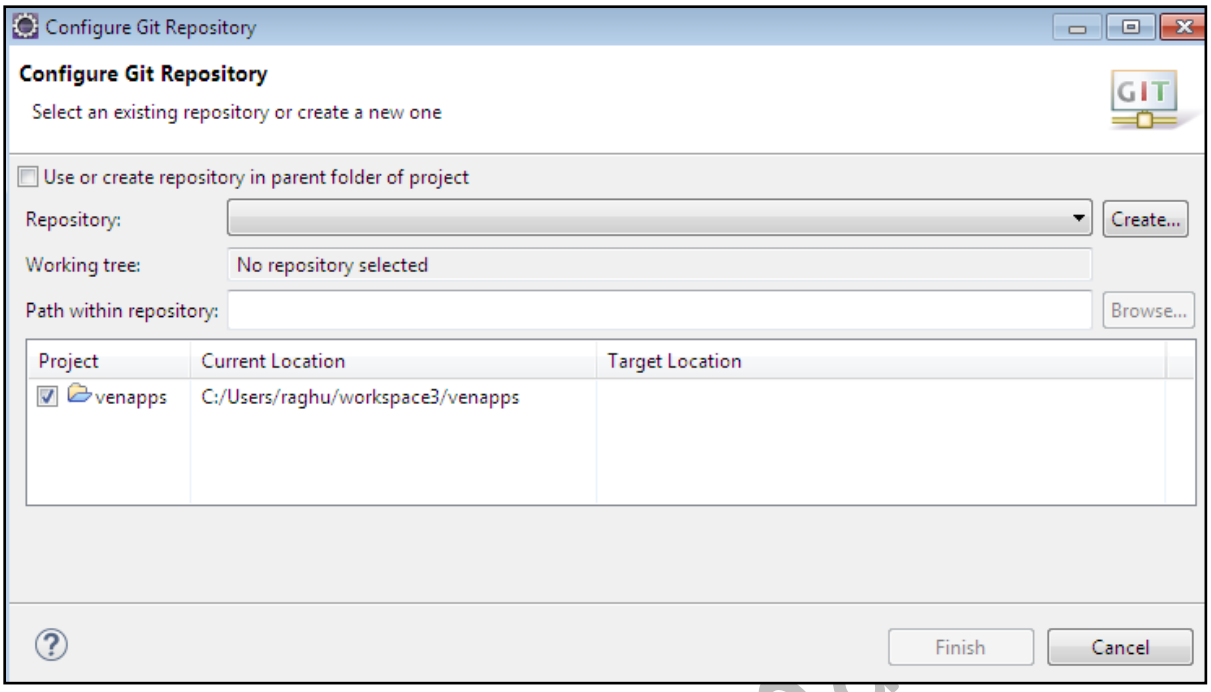


After writing of some code ex: Employee.java

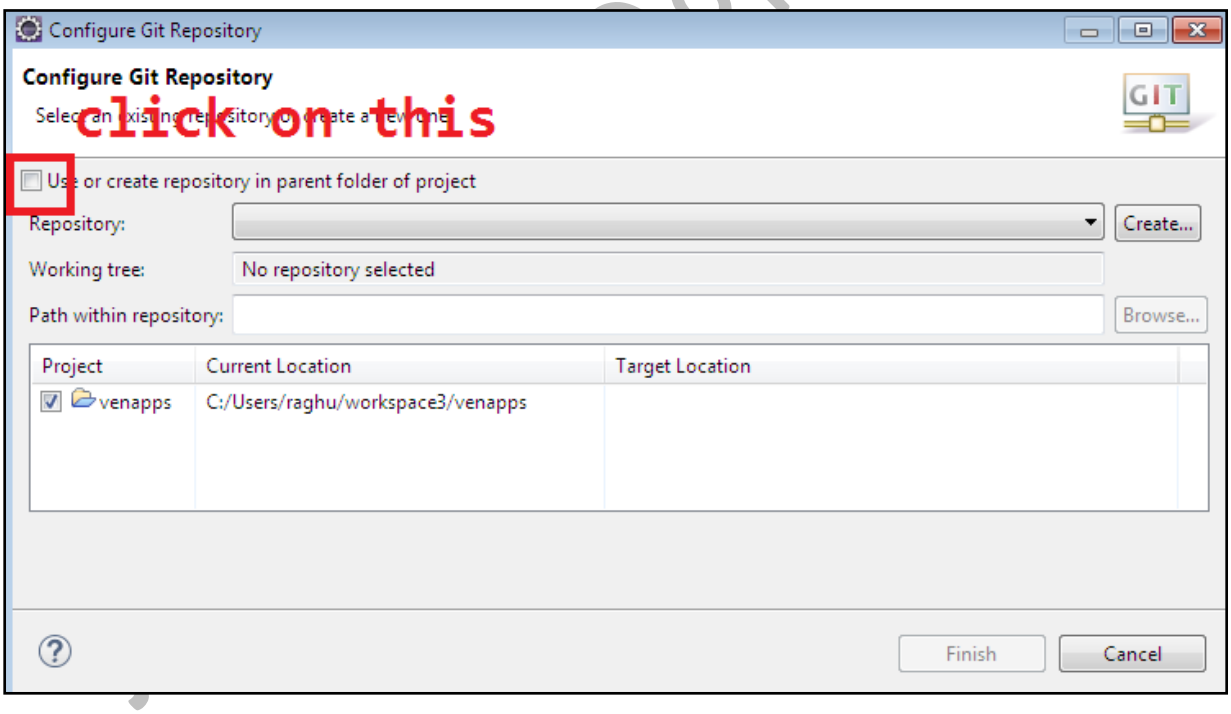
>right click on project > tem > share project



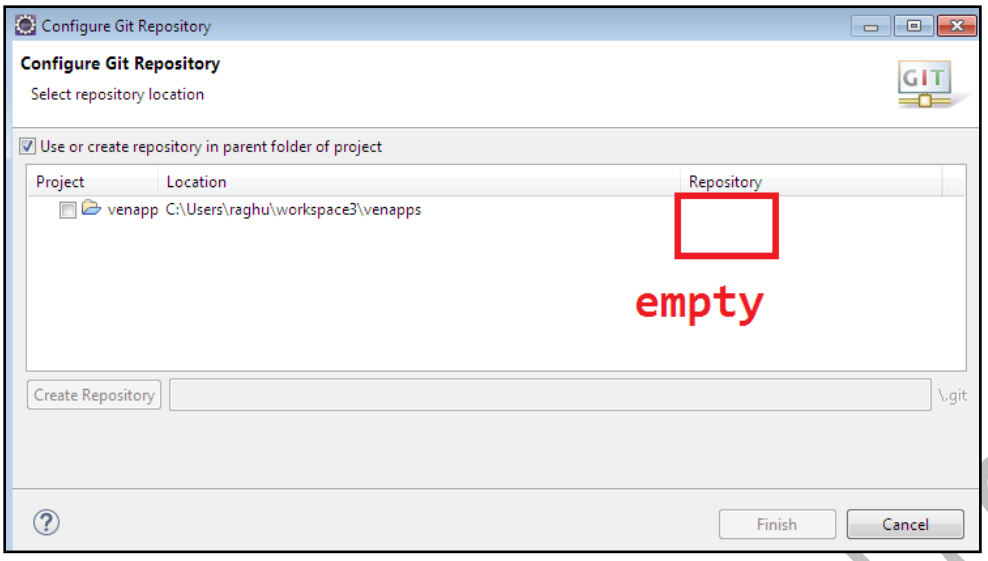
looks as below:

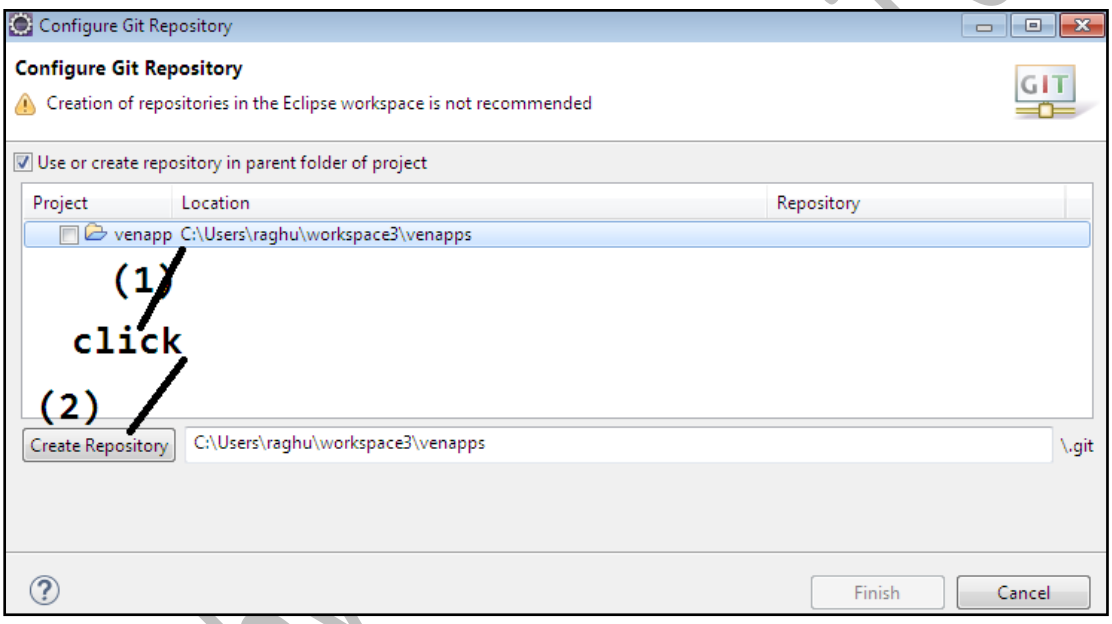


Click on check box:

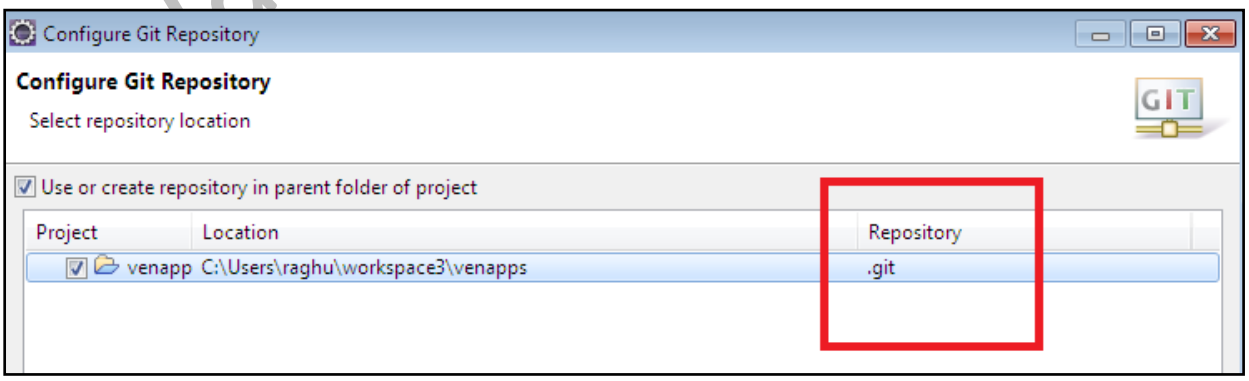


looks as below:





then observe repository name:

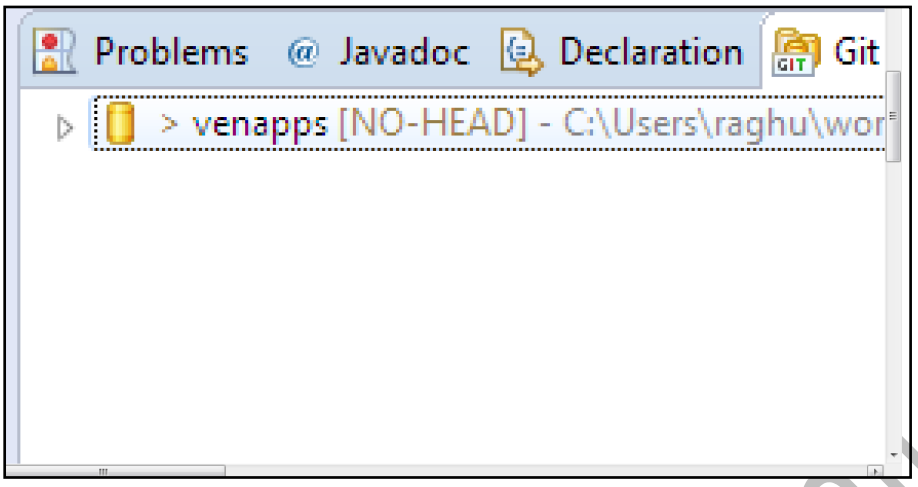


click on finish.

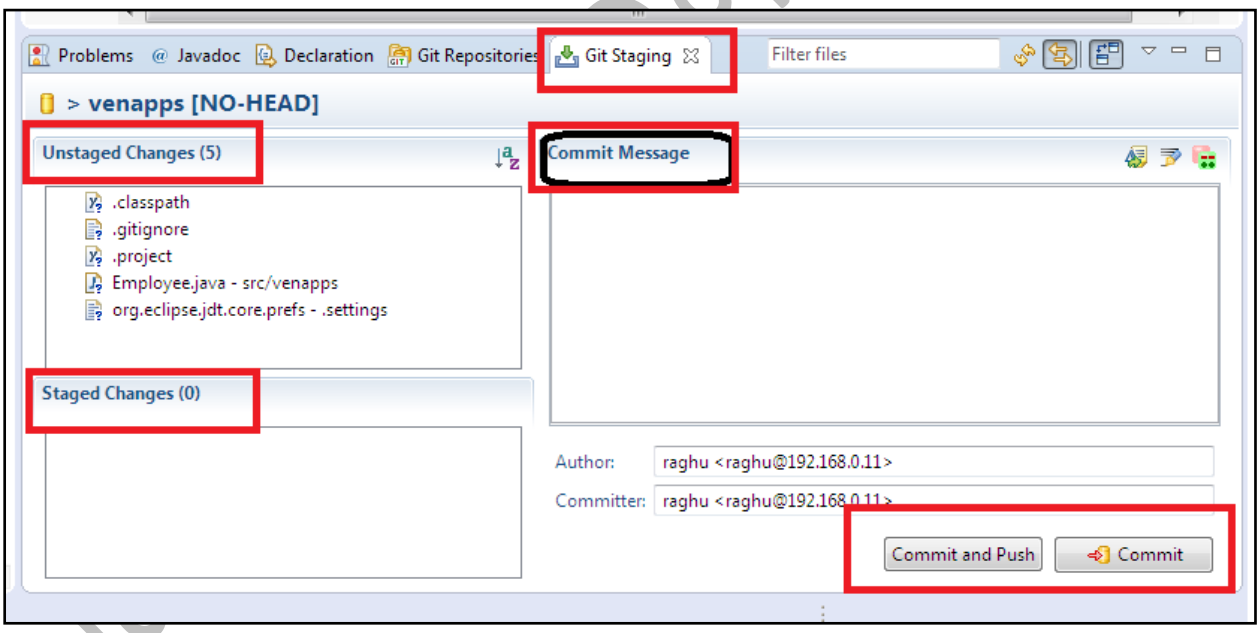
By this we created local and remote repositories.

Now link local and remote, using firsh commit and push.

Come to git repositoes



In staging are:



1)Unstaged changes : files which are not ready to send to local

2)staged chages: file which needs to move to local/remote

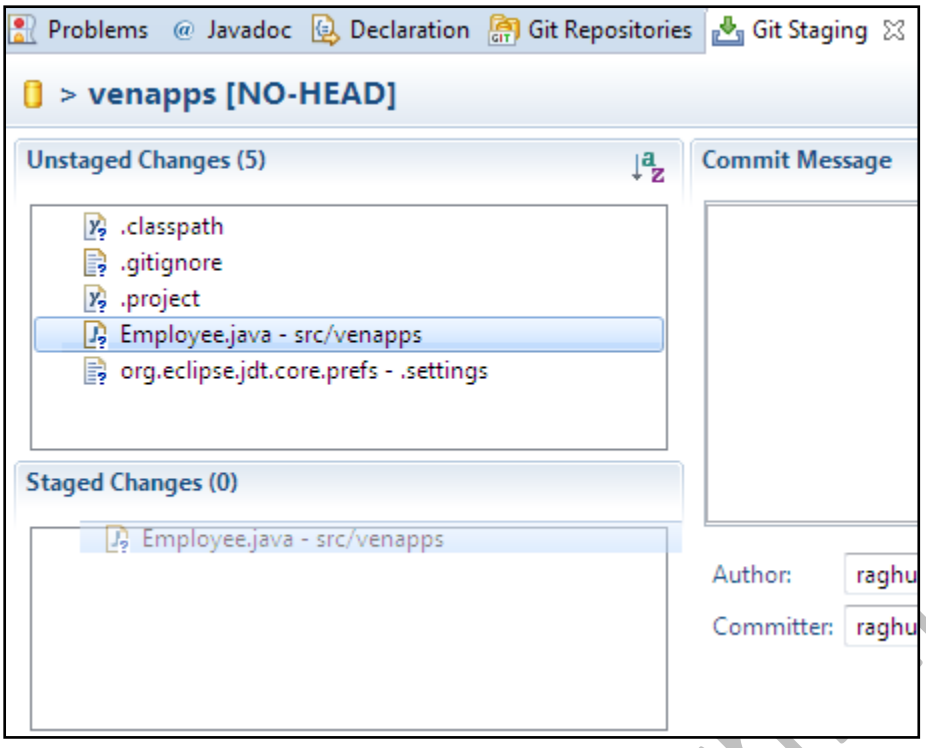
3)commit message: what is the purpose specify here as message for sending of this code.

4) click commit/ commit & push to send the code.

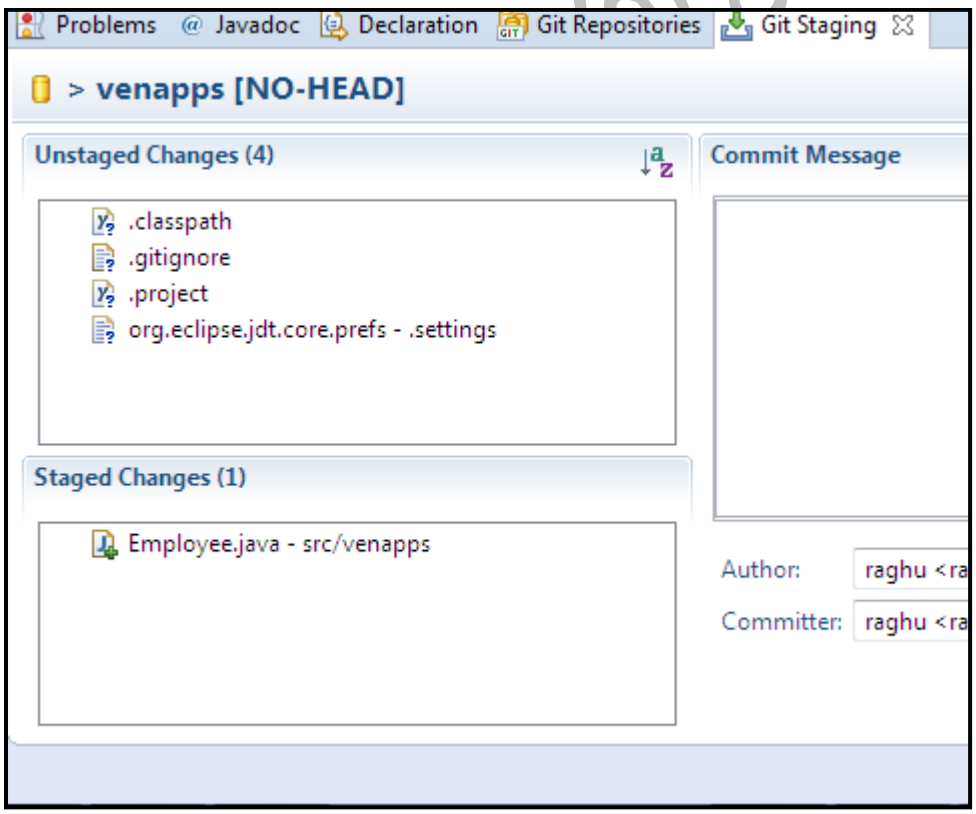
---

*)Click on a file in unstage drag and drop into stage.

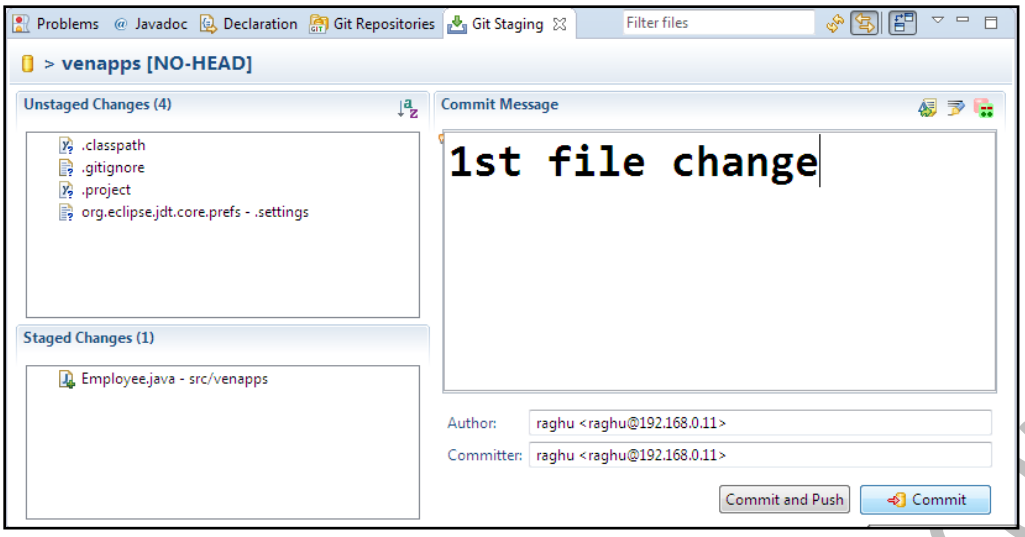
Click and Drag:



After drop

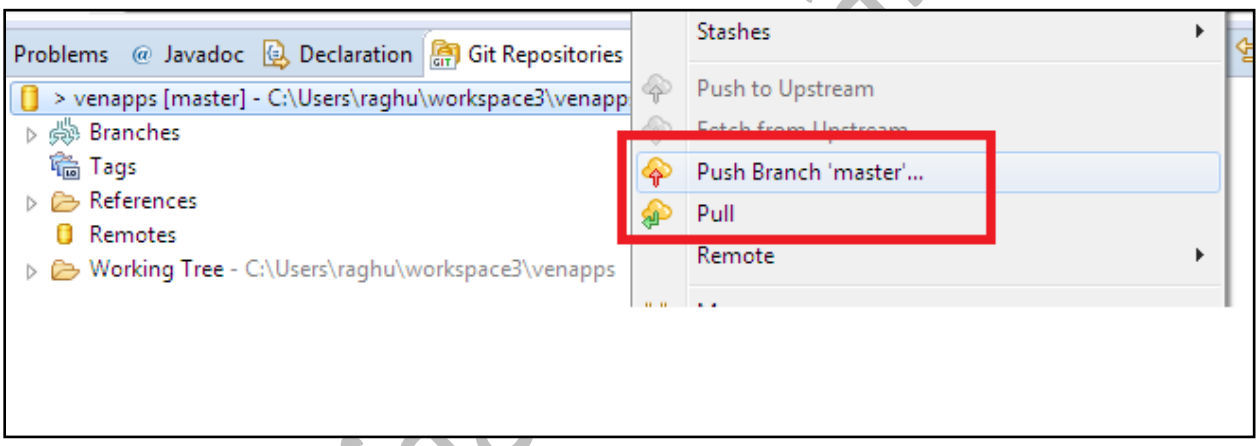


Enter commit message



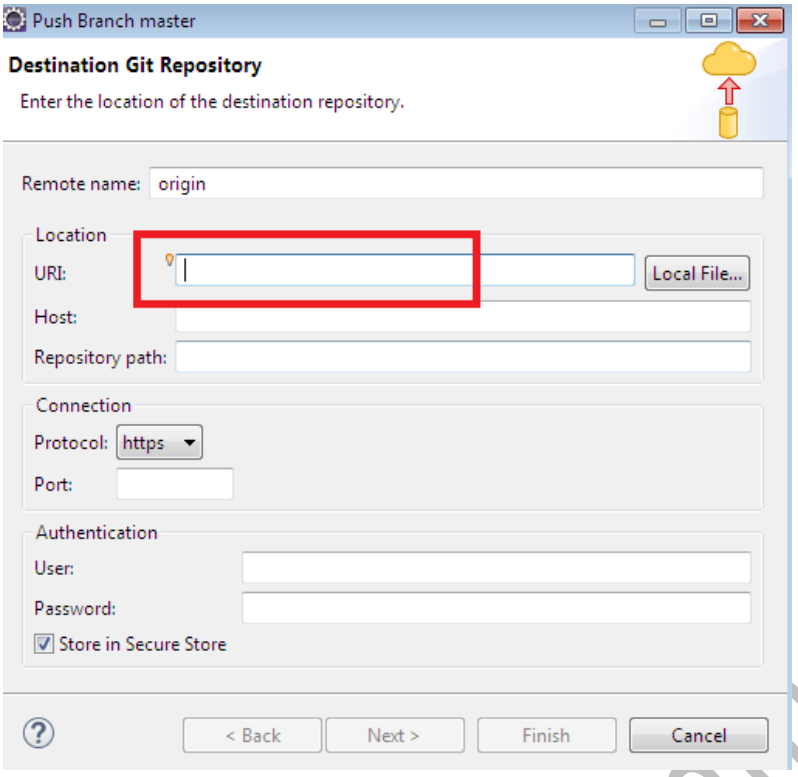
And click on commit.

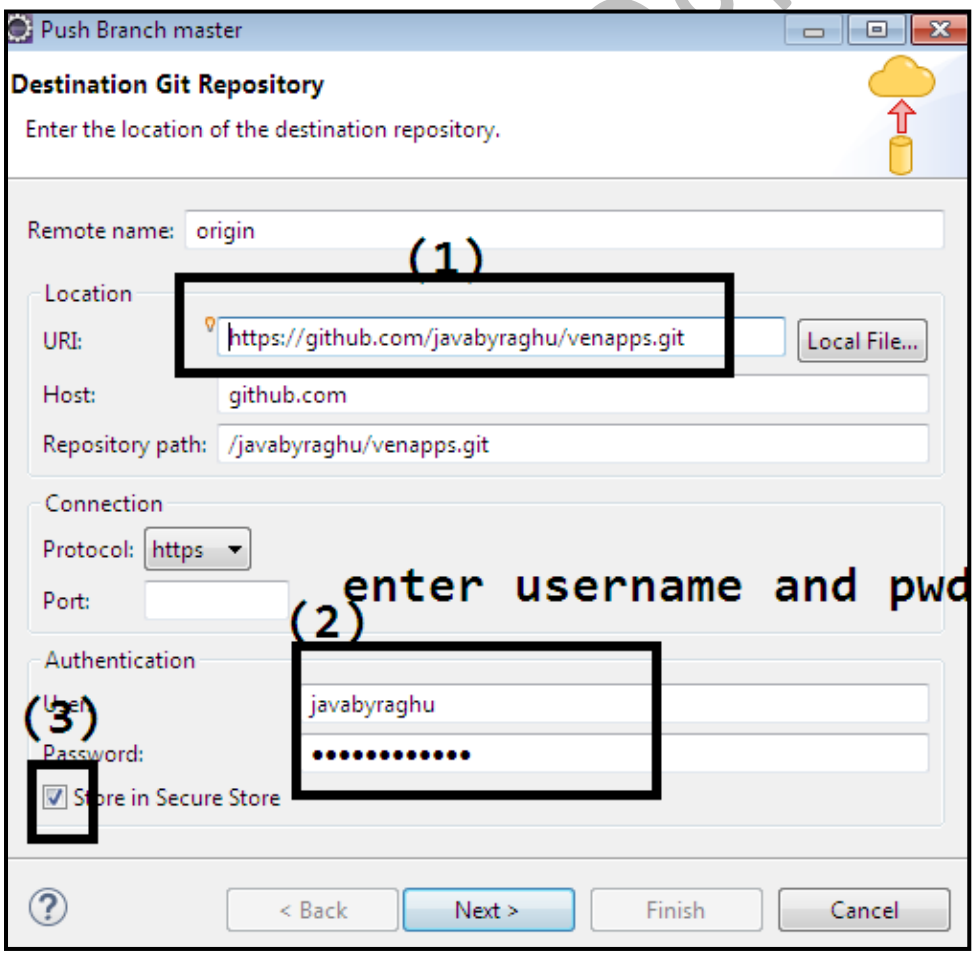
In GitRepository , right click on projectName>choose Push Branch master



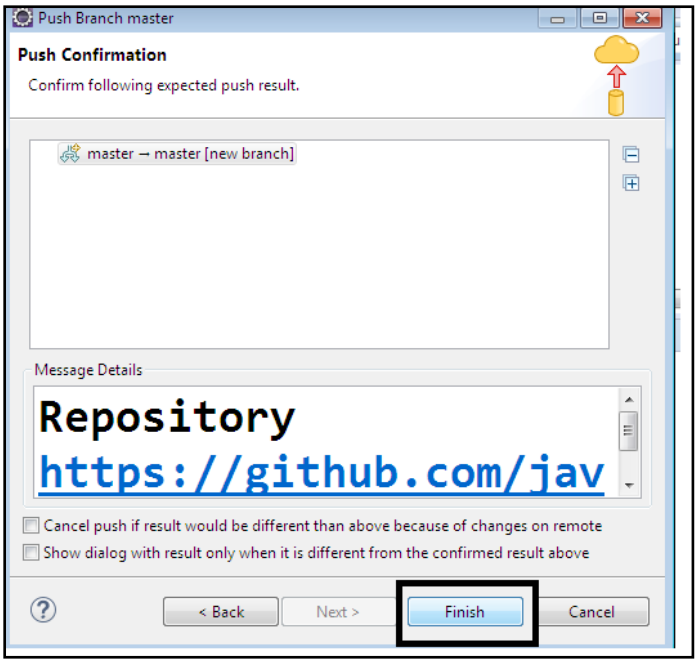
After Push option one screen looks below, there enter URI (Repository link)

Ex: https://github.com/javabyraghu/venapps.git

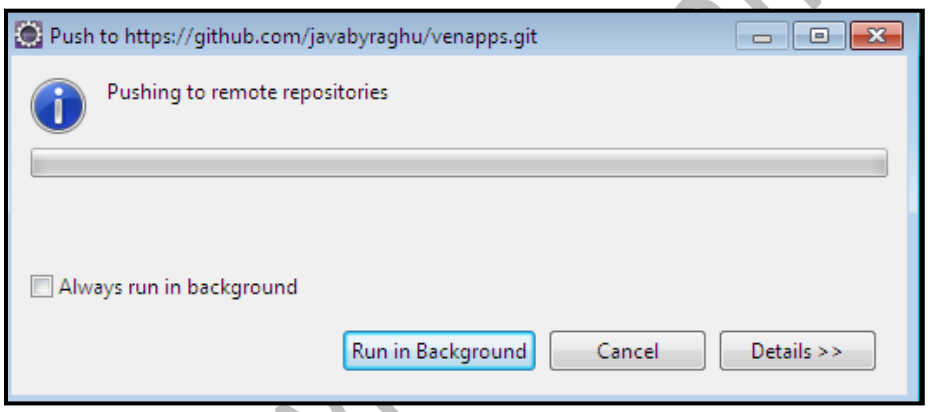




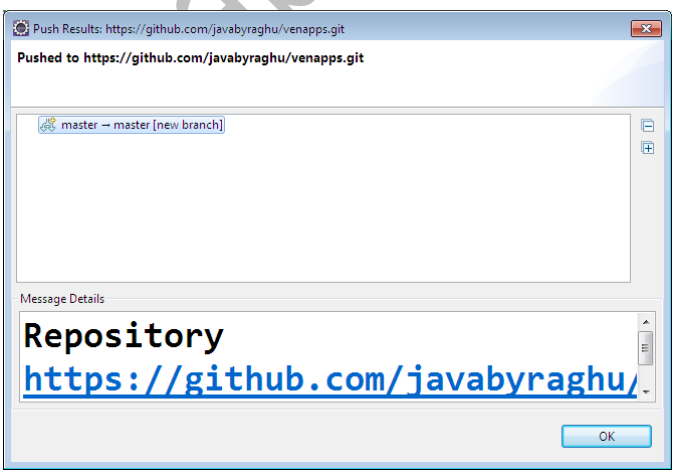
Click on next>next>finish



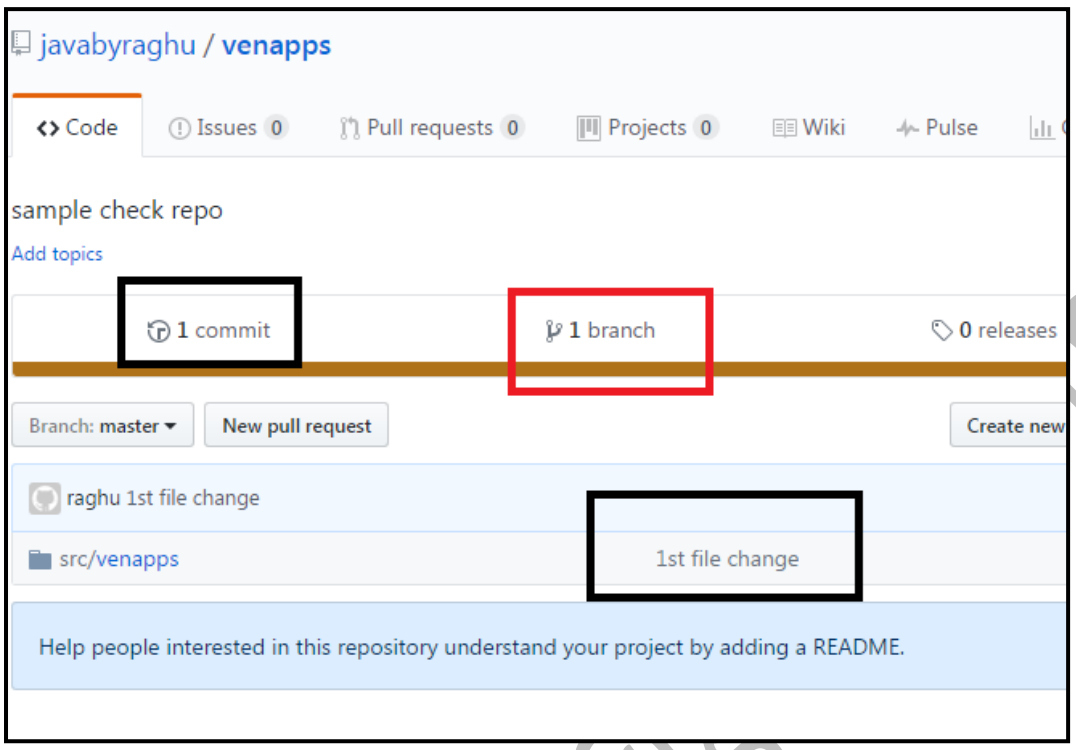
Shows as below:



On success:

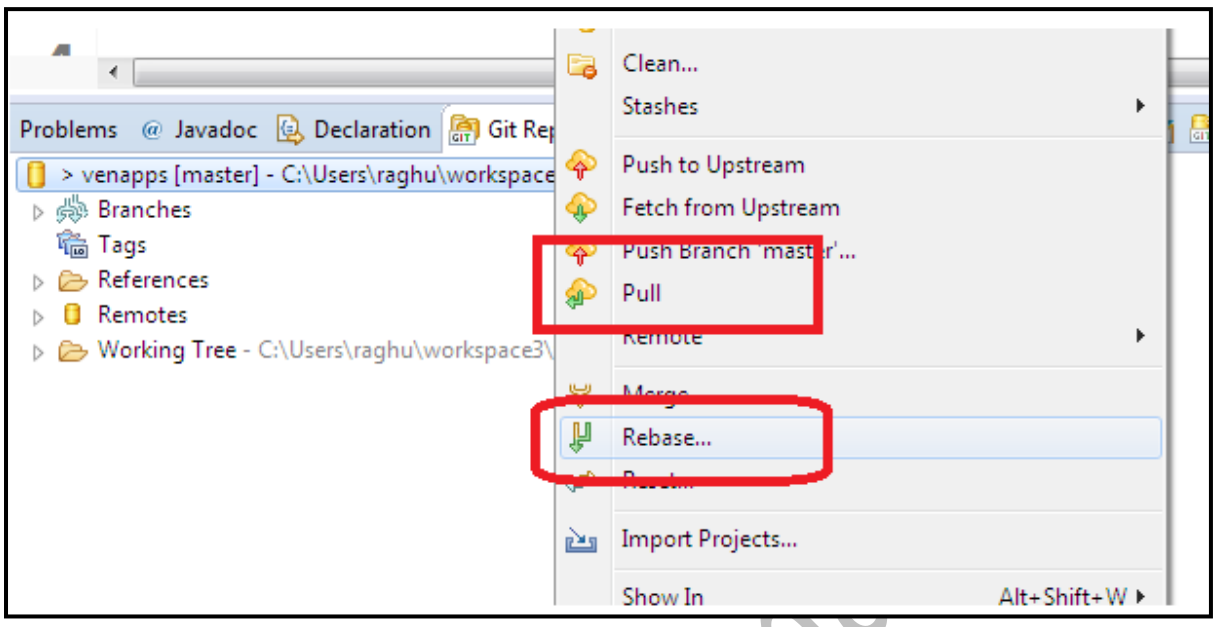


Come back to browser and refresh:

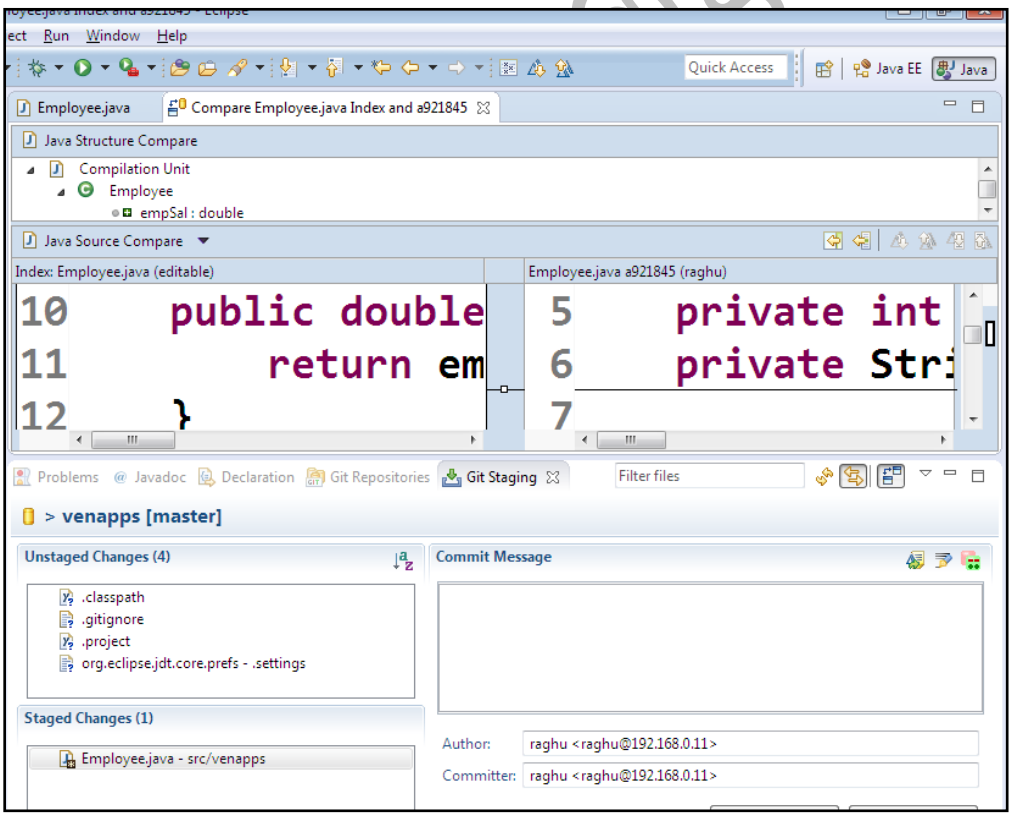


Click on src/venapp>Employee.java

Right click on project: pull and then rebase for other update:

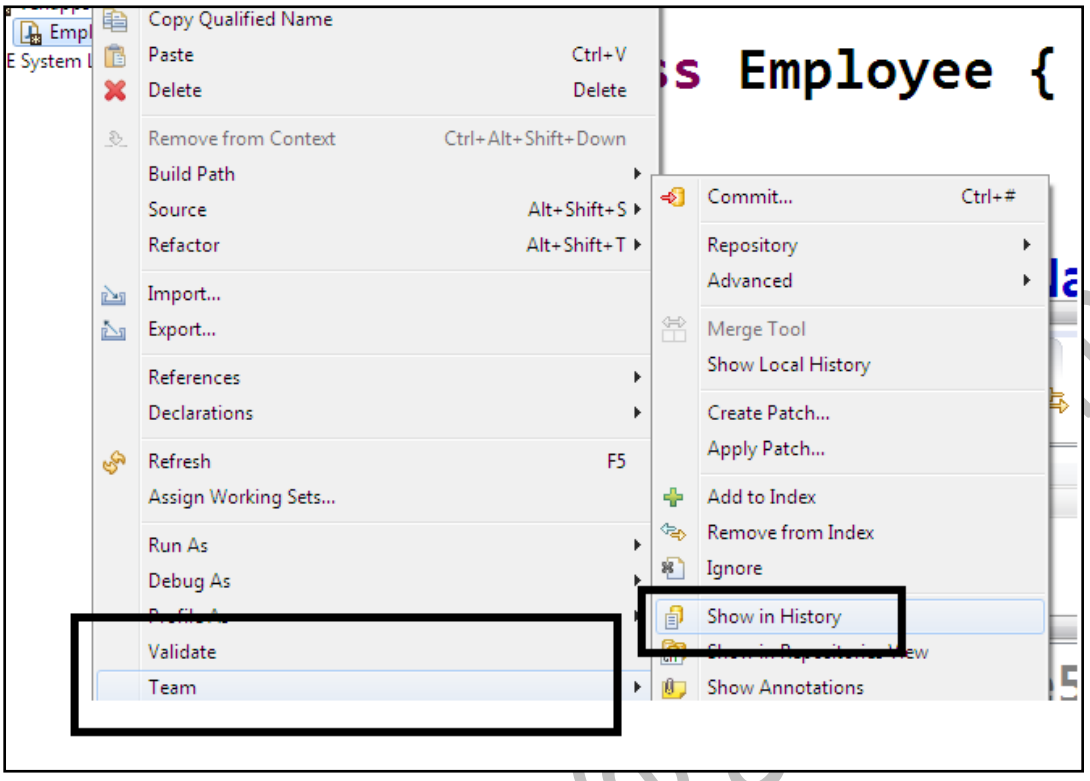


After Ading file to staging ,double click on that to comapre with Old one (previous one)looks as :

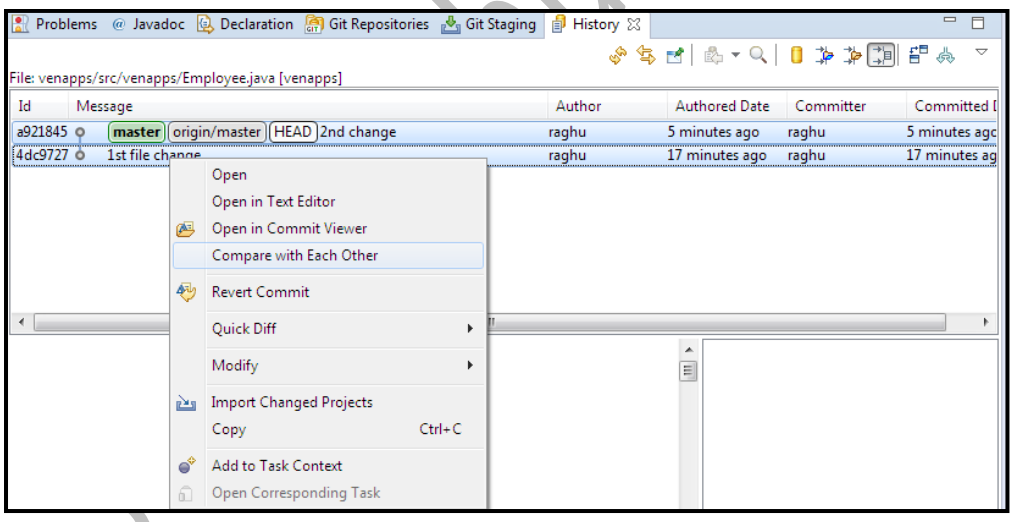


To see history:

Right click on File (ex: Employee.java)> Team>Showin History



View history:



** GIT Merge Conflicts **

If any file code is modified(added/removed) then file version will be changed.

ex: Employee.java = Version#1

1. class Employee
2. {

3.

4. }

Employee.java = Version#2

1. class Employee

2. {

3.+ int empId;

4.+ String empName; - }

5.+ }

V#2 = V#1 [+3,-1].

Git manages version management automatically Here programmer do not need to remember any version number.

** On git pull & rebase, local repository and remote repository versions will become equal.

1. Repositories
2. right click on project
3. "pull" option
4. again right click
5. "rebase" option.

Merge conflict example:

Step#1: Dev#1 has done git pull & rebase

Step#2: Dev#2 has done git pull & rebase

Step#3: Dev#1 modified Employee.java

(that is changed to Version#2) and

did add/commit/push.

Now git has Employee.java Version#2

Step#4: Dev#2 Still working with old code

of Employee.java (Version#1).

File modified and did add/commit/

push, then git shows

Git - Version Conflict: Dev#2

Employee.java(version#1) not

matched with Git Repo Employee.java

(Version#2). Please update before

commit/push.

To Resolve merge conflict steps are.

- project -> right click -> reset
- project -> right click -> pull & rebase
- project -> right click -> merge

Now code will be updated to V#2,

write your code, then >>> add/commit/push.

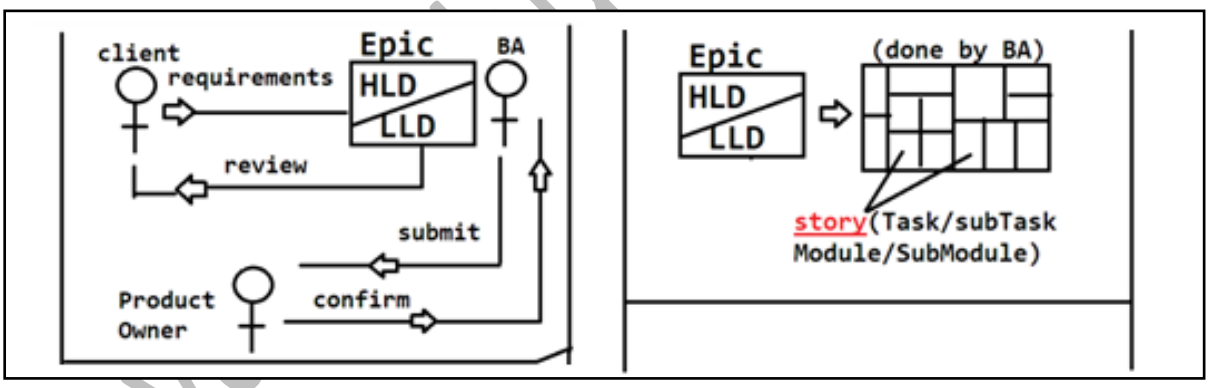
face book group Id: **https://www.facebook.com/groups/thejavatemple/**
email :
javabyraghu@gmail.com

AGILE

It is a SDLC methodology to develop a product in company. It uses Spiral + RAD methods in development, divide and develop parallel. It this process has below Roles :

A. Business Analyst.(BA)
B. Developer (DEV)
C. Quality Analyst.(QA)
D. Architecture (Arch.)

1) Epic: End client/customer provides all his requirements to BA. BA prepares the document (known as EPIC). Which contains functional and technical requirements. It is like a reference book to product/project. We need to develop only points provided in document(not more or less) This document needs a final review by End Client and approval by Product Owner.



2)Epic Conversion: Epic will be divided into 2 parts initially. Those are

- High level Design/Document
- Low level Design/Document

HLDs are constructed for End User(for product understand and usage) and LLDs are done for Programmer View.

3)Scrum Team: To develop the product Owner creates a Team with DEV,BA,QA & Arch with MASTER and Leads.

4)Story creations: A story is a part of project, It can be a module/sub module/task/sub task. A story can be combination of modules/tasks also.

- Size and limits of Stories are defined by BAs.
- Each story Contains "Acceptance Criteria" ie what to implement for given story in point by point to Dev/QA/Arch given by BA.

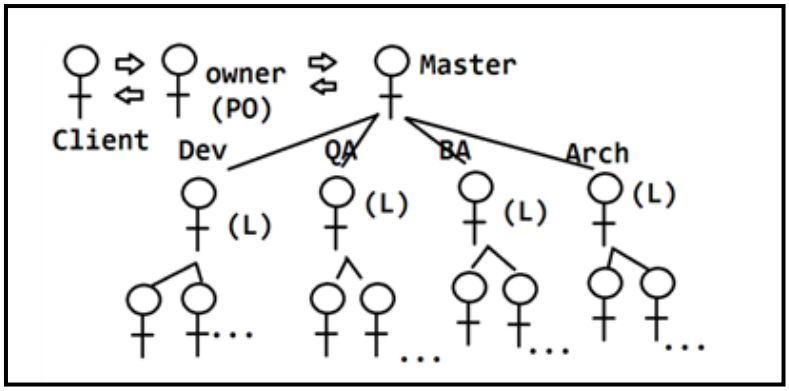
5)Story points Calculations: Time taken to develop a story is known as "Efforts Estimations". These are guess values(Not exact) given by team members.

1 Story point = 8 Hours , 0.5 Story point = 4 Hours

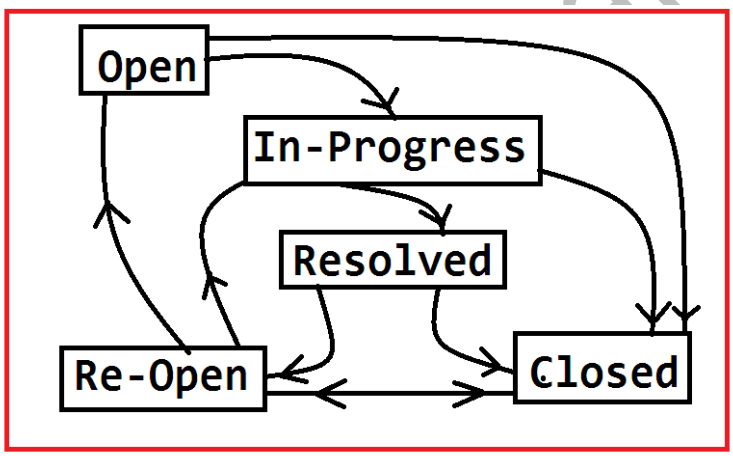
ex: Story-8560 needs 4.5 story points total time (hours)is:36

6) Sprint Planning: "For a period of time what stories we need to implement" is decided in a meeting. Also known as Spring Planning. A sprint is known as milestone/small release/targets. In this meeting BA explains about story to all others. They should understand and provide efforts for those.

- A sprint can be planned for 10 days,15 days, 1 month... (based on company & project)
- At a time we cannot plan for all stories. for example, in our project we have 100 stories and we planned as below:
- Sprint-1 (Story 1 to 12) for 15 days.
- Sprint-2 (Story 13 to 32) for 25 days.
- Unplanned stories comes under product backlogs (Story 33 to 100).
- In Sprint-1 we planned for 12 Stories, but only 10 are finished after 15 days then, 2 will be moved to Product backlogs.



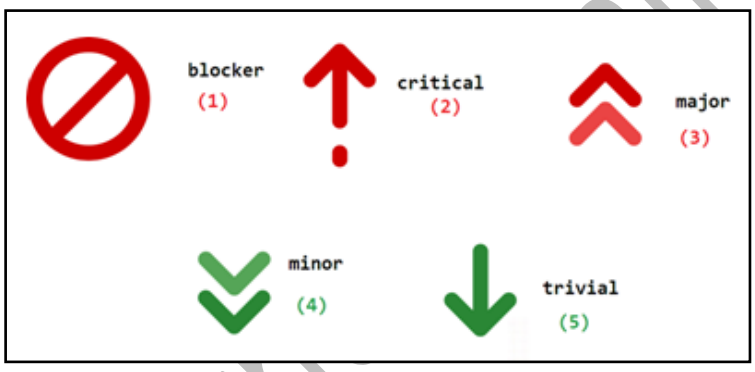
7)Sprint Evaluation: From Day-1 coding/implementation is started by developer. To implement any story dev has to code and Test from his side. To represent work status of a story, we follow Story life cycle. Shown as below, (possible values are: Open->In Progress->Resolved ->Closed or Re-Open)



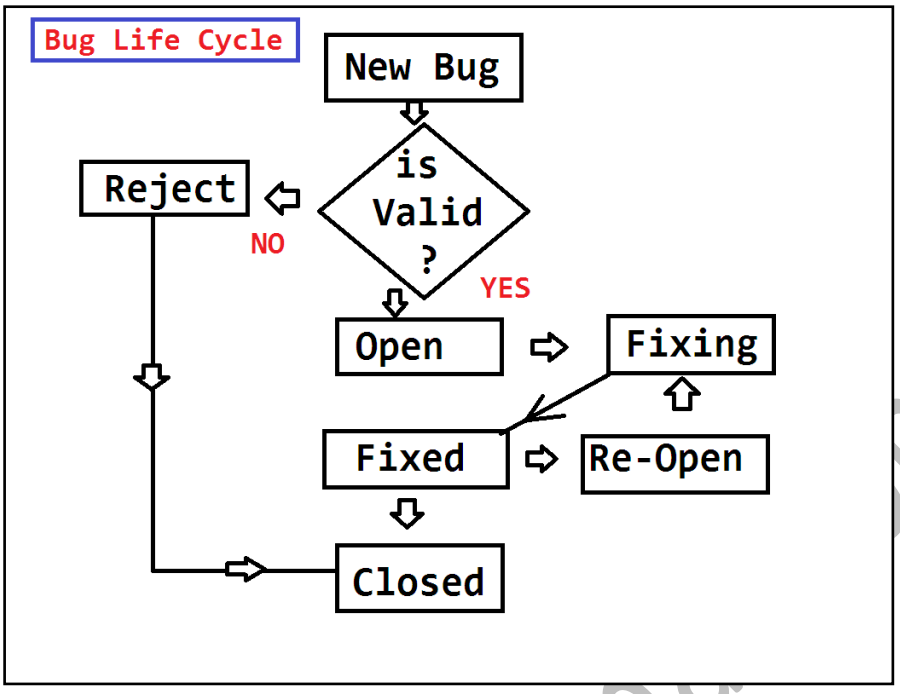
- **Open**: Every developer gets story in Open Status, which indicates ready to start (ready for coding)
- **In-Progress**: Which indicates story is under implementation (coding is started)
- **Resolved**: Indicates Coding and Unit Testing is done. Code is available in Repository (ex: GIT).
- **Closed**: Once Testing done by QA and working fine, then story status will be Closed.
- **Re-Open**: If code contains Bugs, then story is not complete which will be converted to Re-Open and QA creates a BUG with below priority levels.

BUG : It indicates problem in application. Every bug has a level in AGILE process, those are given below with meaning and symbols.

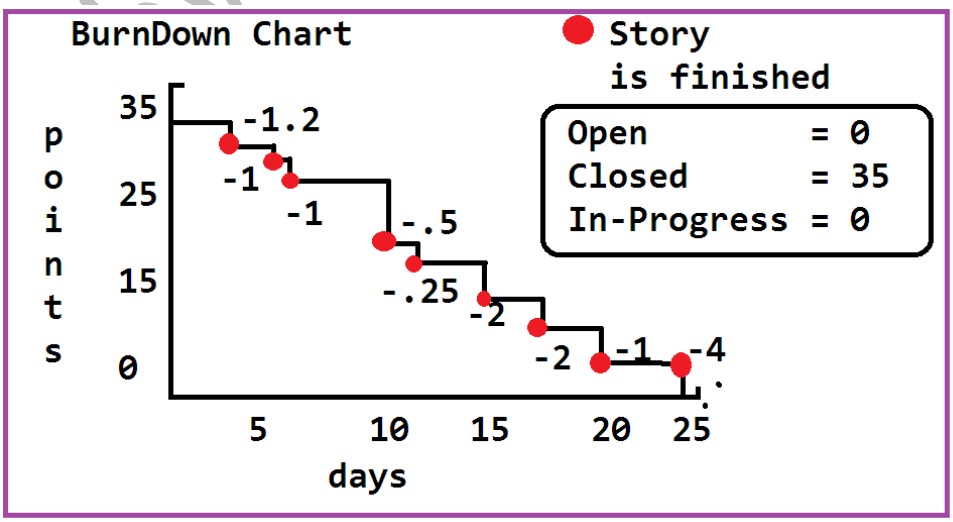
I. **Blocker: Unable to do any process.**
II. **Critical: High level problem (But not stopping complete process)**
III. **Major: Normal/Medium problem (Stopping in one way/ other possible processing ways are available)**
IV. **Minor: Small problem or occurs very rare.**
V. **Trivial: Ignorable problem. not makes any problem in process.**



8)Bug/Defect Life cycle: While developing a story, it may contain problems (bug) or it is incomplete then QA identifies the bug and reports to Developer. ex: Bug-32(Critical): Email is not sending while vendor registration. Attached to Story-54(Vendor Registration). Developer checks bug and validates , if valid starts fixing else reject the bug.



9)Burn Down Chart: This is a global chart, constructed to represent status of the sprint/Sprints. Up-to-date what % of work done is shown by this chart. If one story is finished, then it shows -value ex: -3.25 points. At end all stories should be finished. +ve value indicates Work is re-opened.



10)Scrum Meetings: On Starting Sprint, every day we need to provide status updates to Scrum Master. For this every day meeting is taken by master mostly 15-20 minutes. ex: 10:45 to 11:00 AM. sometimes 3:00 to 3:25 PM. In this meeting we need to tell "what we did yester day, what we will do today and tomorrow".

FB Group: **https://www.facebook.com/groups/thejavatemple**

ECLIPSE SHORTCUTS

1. Manage Files and Projects	
Ctrl+N	Create new project using the Wizard
Ctrl+Alt+N	Create new project, file, class, etc.
Alt+F, then .	Open project, file, etc.
Ctrl+Shift+R	Open Resource (file, folder or project)
Alt+Enter	Show and access file properties
Ctrl+S	Save current file
Ctrl+Shift+S	Save all files
Ctrl+W	Close current file
Ctrl+Shift+W	Close all files
F5	Refresh content of selected element with local file system

2. Editor Window	
Focus/ cursor must be in Editor Window for these to work.	
F12	Jump to Editor Window
Ctrl+Page Down/Ctrl+Page Up	Switch to next editor / switch to previous editor
Ctrl+M	Maximize or un-maximize current Editor Window (also works for other Windows)
Ctrl+E	Show list of open Editors. Use arrow keys and enter to switch
Ctrl+F6/Ctrl+Shift+F6	Show list of open Editors. Similar to ctrl+e but switches immediately upon release of ctrl
Alt+Arrow Left/Alt+Arrow Right	Go to previous / go to next Editor Window
Alt+-	Open Editor Window Option menu
Ctrl+F10	Show view menu (features available on left

	vertical bar: breakpoints, bookmarks, line numbers, …)
Ctrl+F10, then n	Show or hide line numbers
Ctrl+Shift+Q	Show or hide the diff column on the left (indicates changes since last save)

3. Navigate in Editor	
Home/End	Jump to beginning / jump to end of indention. Press home twice to jump to beginning of line
Ctrl+Home/End	Jump to beginning / jump to end of source
Ctrl+Arrow Right/Arrow Left	Jump one word to the left / one word to the right
Ctrl+Shift+Arrow Down/Arrow Up	Jump to previous / jump to next method
Ctrl+L	Jump to Line Number. To hide/show line numbers, press ctrl+F10 and select 'Show Line Numbers'
Ctrl+Q	Jump to last location edited
Ctrl+./Ctrl+,	Jump to next / jump to previous compiler syntax warning or error
Ctrl+Shift+P	With a bracket selected: jump to the matching closing or opening bracket
Ctrl+[+]/Ctrl+- on numeric keyboard	Collapse / Expand current method or class
Ctrl+[/]/Ctrl+* on numeric keyboard	Collapse / Expand all methods or classes
Ctrl+Arrow Down/Ctrl+Arrow Up	Scroll Editor without changing cursor position
Alt+Page Up/Alt+Page Down	Next Sub-Tab / Previous Sub-Tab

4. Select Text	
Shift+Arrow Right/Arrow Left	Expand selection by one character to the left / to the right
Ctrl+Shift+Arrow Right/Arrow Left	Expand selection to next / previous word

Shift+Arrow Down/Arrow Up	Expand selection by one line down / one line up
Shift+End/Home	Expand selection to end / to beginning of line
Ctrl+A	Select all
Alt+Shift+Arrow Up	Expand selection to current element (e.g. current one-line expression or content within brackets)
Alt+Shift+Arrow Left/Arrow Right	Expand selection to next / previous element
Alt+Shift+Arrow Down	Reduce previously expanded selection by one step

5. Edit Text	
Ctrl+C/Ctrl+X/Ctrl+V	Cut, copy and paste
Ctrl+Z	Undo last action
Ctrl+Y	Redo last (undone) action
Ctrl+D	Delete Line
Alt+Arrow Up/Arrow Down	Move current line or selection up or down
Ctrl+Alt+Arrow Up/Ctrl+Alt+Arrow Down/	Duplicate current line or selection up or down
Ctrl+Delete	Delete next word
Ctrl+Backspace	Delete previous word
Shift+Enter	Enter line below current line
Shift+Ctrl+Enter	Enter line above current line
Insert	Switch between insert and overwrite mode
Shift+Ctrl+Y	Change selection to all lower case
Shift+Ctrl+X	Change selection to all upper case

6. Search and Replace	
Ctrl+F	Open find and replace dialog
Ctrl+K/Ctrl+Shift+K	Find previous / find next occurrence of search

	term (close find window first)
Ctrl+H	Search Workspace (Java Search, Task Search, and File Search)
Ctrl+J/Ctrl+Shift+J	Incremental search forward / backwards. Type search term after pressing ctrl+j, there is now search window
Ctrl+Shift+O	Open a resource search dialog to find any class

7. Indentions and Comments	
Tab/Shift+Tab	Increase / decrease indent of selected text
Ctrl+I	Correct indention of selected text or of current line
Ctrl+Shift+F	Autoformat all code in Editor using code formatter
Ctrl+/	Comment / uncomment line or selection (adds '//')
Ctrl+Shift+/	Add Block Comment around selection (adds '/... */')
Ctrl+Shift+	Remove Block Comment
Alt+Shift+J	Add Element Comment (adds '/** ... */')

8. Editing Source Code	
Ctrl+Space	Opens Content Assist (e.g. show available methods or field names)
Ctrl+1	Open Quick Fix and Quick Assist
Alt+/	Propose word completion (after typing at least one letter). Repeatedly press alt+/ until reaching correct name
Ctrl+Shift+Insert	Deactivate or activate Smart Insert Mode (automatic indention, automatic brackets, etc.)

9. Code Information

Ctrl+O	Show code outline / structure
F2	Open class, method, or variable information (tooltip text)
F3	Open Declaration: Jump to Declaration of selected class, method, or parameter
F4	Open Type Hierarchy window for selected item
Ctrl+T	Show / open Quick Type Hierarchy for selected item
Ctrl+Shift+T	Open Type in Hierarchy
Ctrl+Alt+H	Open Call Hierarchy
Ctrl+Shift+U	Find occurrences of expression in current file
Ctrl+move over method	Open Declaration or Implementation

10. Refactoring	
Alt+Shift+R	Rename selected element and all references
Alt+Shift+V	Move selected element to other class or file (With complete method or class selected)
Alt+Shift+C	Change method signature (with method name selected)
Alt+Shift+M	Extract selection to method
Alt+Shift+L	Extract local variable: Create and assigns a variable from a selected expression
Alt+Shift+I	Inline selected local variables, methods, or constants if possible (replaces variable with its declarations/ assignment and puts it directly into the statements)

11. Run and Debug	
Ctrl+F11	Save and launch application (run)
F11	Debug
F5	Step Into function

F6	Next step (line by line)
F7	Step out
F8	Skip to next Breakpoint

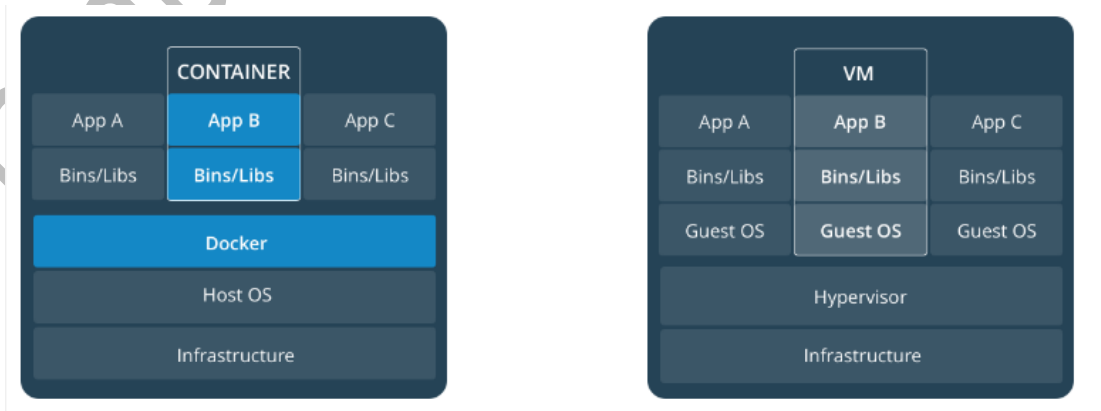
12. The Rest	
Ctrl+F7/Ctrl+Shift+F7	Switch forward / backward between views (panels). Useful for switching back and forth between Package Explorer and Editor.
Ctrl+F8/Ctrl+Shift+F8	Switch forward / backward between perspectives
Ctrl+P	Print
F1	Open Eclipse Help
Shift+F10	Show Context Menu right click with mouse

DOCKER

Docker is a platform for developers and sys admins to build, share, and run applications with containers. The use of containers to deploy applications is called containerization. Containers are not new, but their use for easily deploying applications is.

Containerization is increasingly popular because containers are:

1. Flexible: Even the most complex applications can be containerized.
2. Lightweight: Containers leverage and share the host kernel, making them much more efficient in terms of system resources than virtual machines.
3. Portable: You can build locally, deploy to the cloud, and run anywhere.
4. Loosely coupled: Containers are highly self sufficient and encapsulated, allowing you to replace or upgrade one without disrupting others.
5. Scalable: You can increase and automatically distribute container replicas across a datacenter.
6. Secure: Containers apply aggressive constraints and isolations to processes without any configuration required on the part of the user.



Containers and virtual machines:-

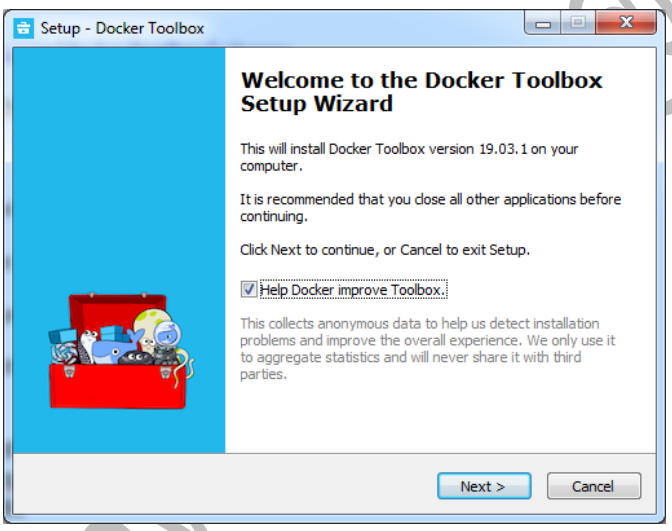
A container runs natively on Linux and shares the kernel of the host machine with other containers. It runs a discrete process, taking no more memory than any other executable, making it lightweight.

By contrast, a virtual machine (VM) runs a full-blown "guest" operating system with virtual access to host resources through a hypervisor. In general, VMs incur a lot of overhead beyond what is being consumed by your application logic.

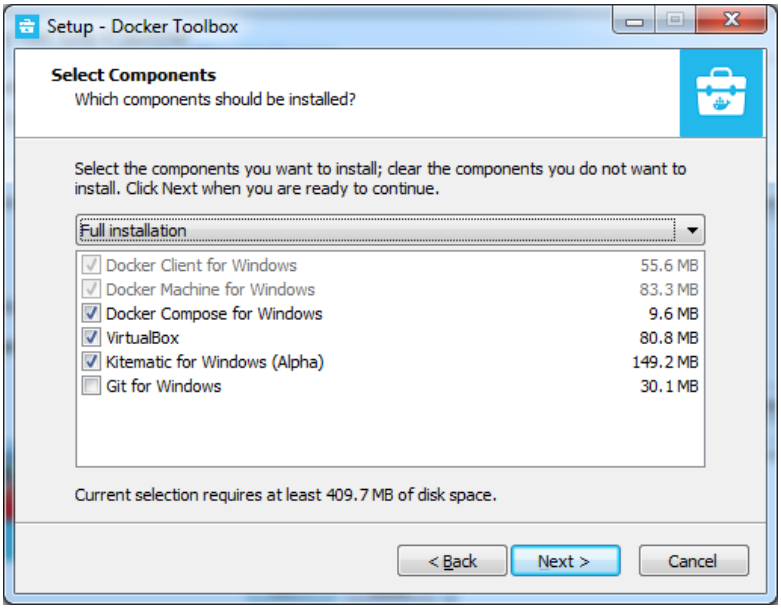
STEP#1 DOCKER DOWNLOAD AND INSTALL

https://github.com/docker/toolbox/releases

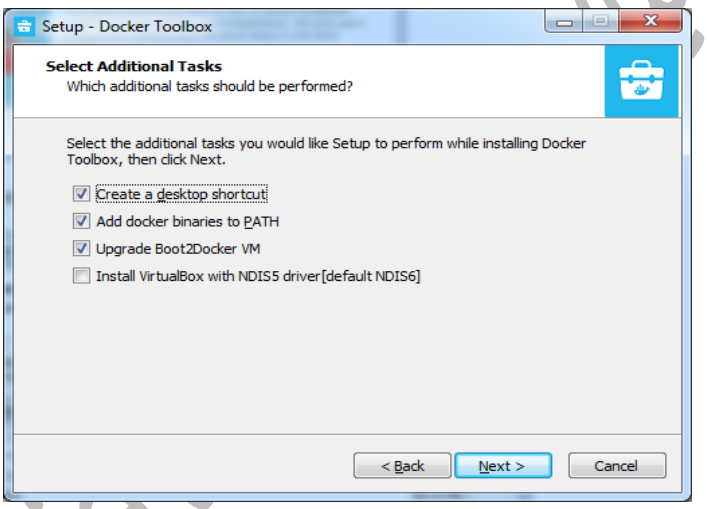
→ **Download Docker → double click on setup file → Next**



→ **Choose Full Installation**



→ **Click on Next**



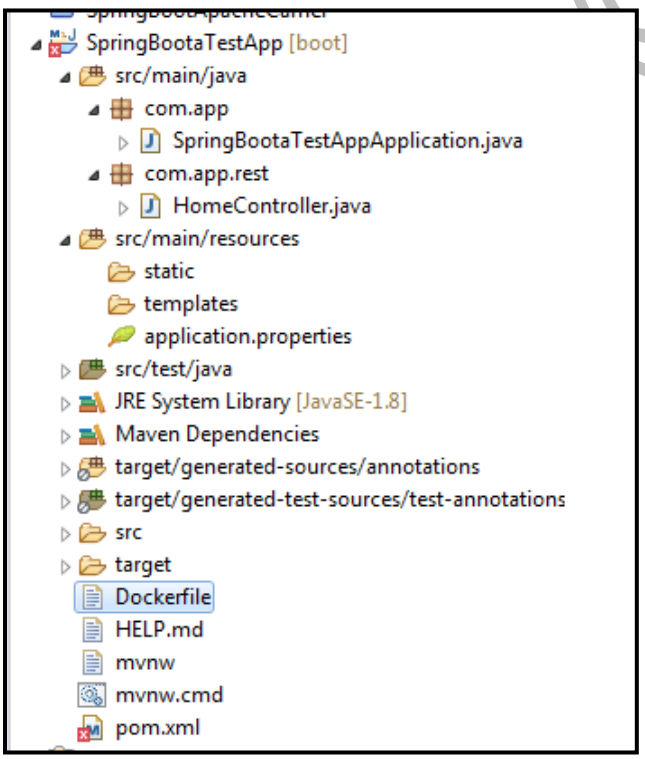
→ **Next → Install**

STEP#2 CREATE ONE SPRING BOOT PROJECT WITH ONE REST CONTROLLER

```
package com.app.rest;
```

```
import org.springframework.web.bind.annotation.GetMapping;
import
org.springframework.web.bind.annotation.RestController;

@RestController
public class HomeController {

	@GetMapping("/reg")
	public String show() {
		return "Home";
	}
}
```

→ **Create File with name : Dockerfile Under Project folder**



→ **Right click on project**

→ **Run As → Maven Install (it generates Jar file under target folder)**

→ add content to docker file (example given below)

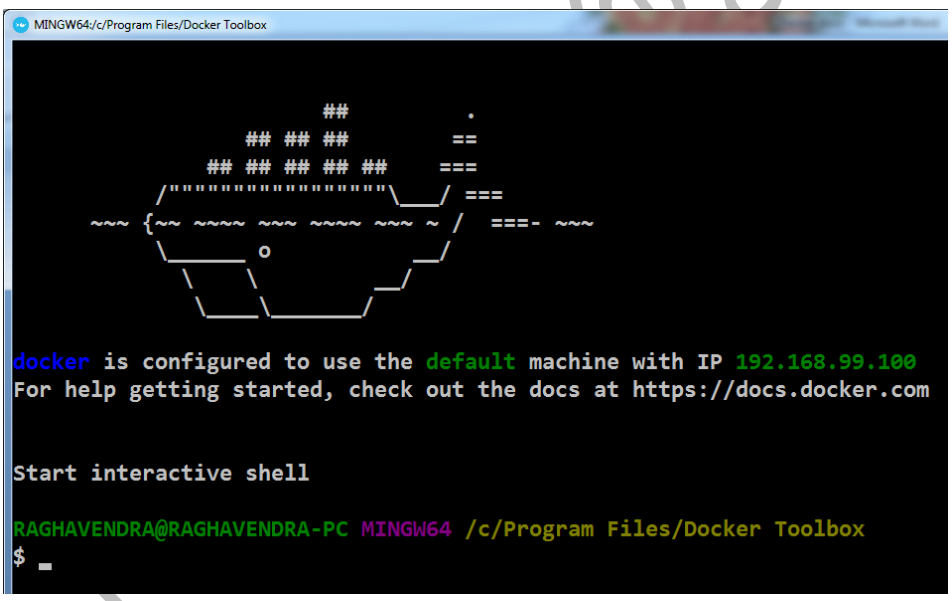
Dockerfile

```
FROM openjdk:8
VOLUME /tmp
EXPOSE 8080
ADD target/springboot-dockerapp.jar springboot-dockerapp.jar
ENTRYPOINT ["java","-jar","/springboot-dockerapp.jar"]
```

STEP#3 START DOCKER

→ Click on Docker Docker Quickstart Terminal Icon

→ wait for few minutes (first time takes time 10-20 mins)



→ move to Project root folder in docker.

Ex: if Project is created in STS under "D:\bootworkspace_sts4_cloud_2.1.6\SpringBootaTestApp" then type command as:

$ cd D:/bootworkspace_sts4_cloud_2.1.6/SpringBootaTestApp

```
RAGHAVENDRA@RAGHAVENDRA-PC MINGW64 /c/Program Files/Docker Toolbox
$ cd D:/bootworkspace_sts4_cloud_2.1.6/SpringBootaTestApp

RAGHAVENDRA@RAGHAVENDRA-PC MINGW64 /d/bootworkspace_sts4_cloud_2.1.6/SpringBoota
TestApp
$
```

STEP#4 CREATE DOCKER CONTAINER IMAGE

docker build -f Dockerfile -t <AnyImageName> .

Here dot(.) indicates current location

$ docker build -f Dockerfile -t springboot-dockerapp .

```
$ docker build -f Dockerfile -t springboot-dockerapp .
Sending build context to Docker daemon  16.97MB
Step 1/4 : FROM openjdk:8
 ---> 1af102cad8ba
Step 2/4 : EXPOSE 8080
 ---> Using cache
 ---> 240f20b8f653
Step 3/4 : ADD target/spring-app.jar spring-app.jar
 ---> Using cache
 ---> 4107df1164bb
Step 4/4 : ENTRYPOINT ["java","-jar","/spring-app.jar"]
 ---> Using cache
 ---> e95300381a8a
Successfully built e95300381a8a
Successfully tagged springboot-dockerapp:latest
SECURITY WARNING: You are building a Docker image from Windows against a non-Win
dows Docker host. All files and directories added to build context will have '-r
wxr-xr-x' permissions. It is recommended to double check and reset permissions f
or sensitive files and directories.
```

→ **Check for list of images created in docker**

$ docker image ls

→ **Run Docker Image**

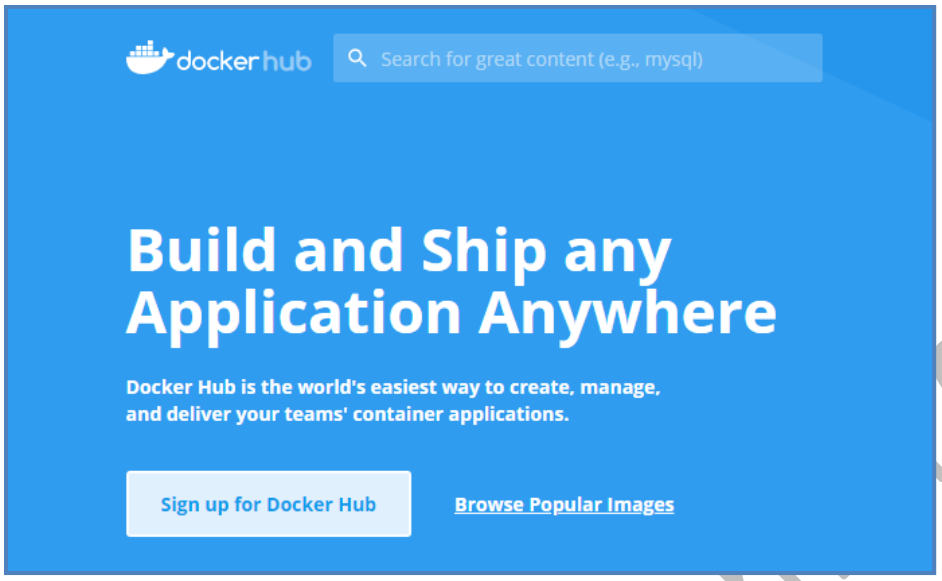
$ docker run -p 9090:8080 springboot-dockerapp

> Goto browser and enter URL:

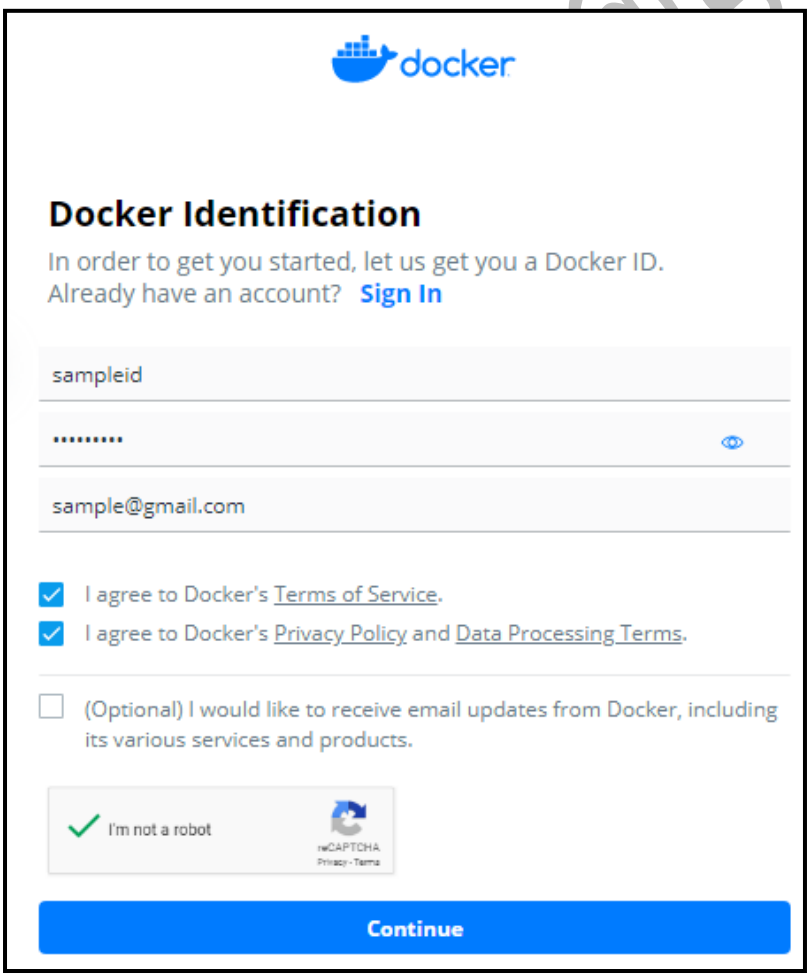
Example URL: http://192.168.99.100:9090/show

STEP#5 CREATE ACCOUNT IN DOCKER HUB

→ URL : https://hub.docker.com/ → **Choose sing up option and register once**



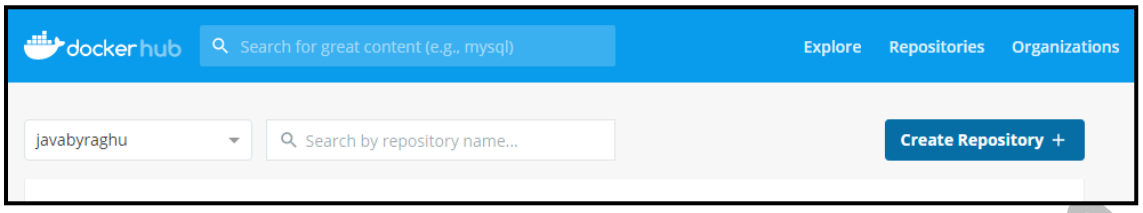
→ **Enter details here**



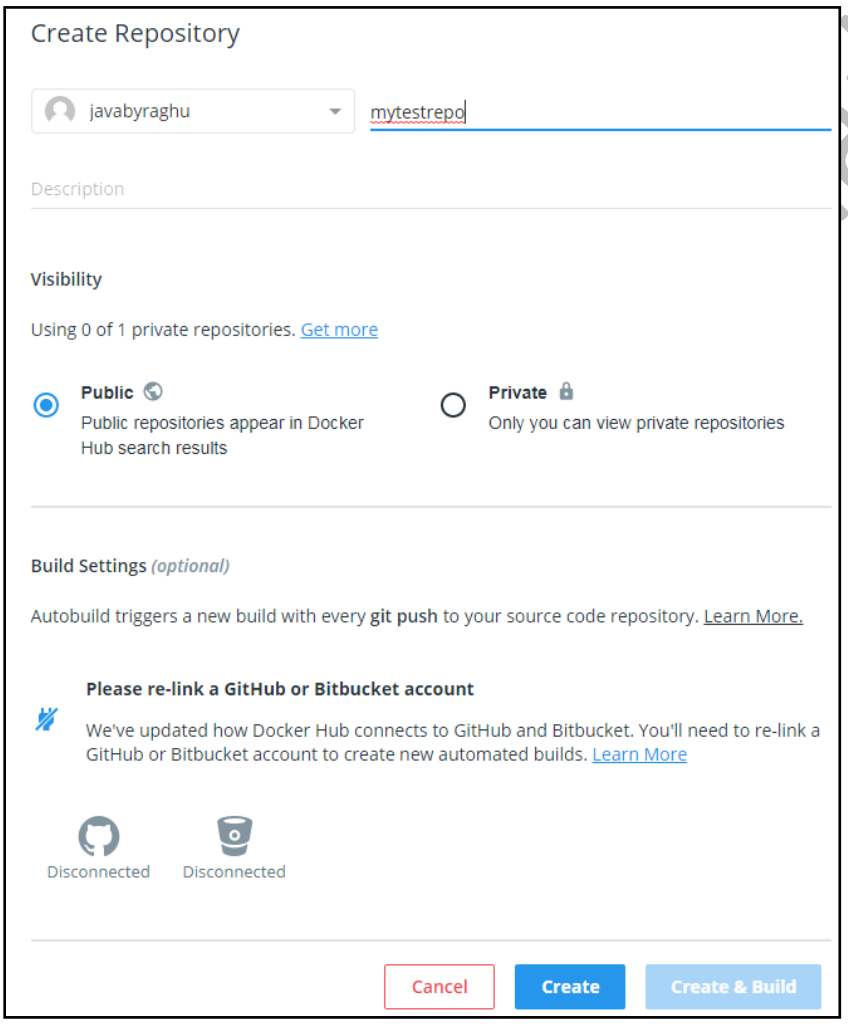
→ **Verify Email id at your inbox**

STEP#6 CREATE REPOSITORY HERE

→ Click on Button Create Repository



→ Enter repository name (Ex : mytestrepo) and click on create button



→ See confirmation screen with sample push command

STEP#7 LOGIN IN DOCKER TERMINAL

→ come back to docker terminal → login here

$docker login

UserName : docker account username

password: docker password

```
RAGHAVENDRA@RAGHAVENDRA-PC MINGW64 /c/Program Files/Docker To
$ docker login
Login with your Docker ID to push and pull images from Docker
have a Docker ID, head over to https://hub.docker.com to crea
Username: javabyraghu
Password:
WARNING! Your password will be stored unencrypted in C:\Users
r\config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#c

Login Succeeded
```

STEP#8 CREATE NEW TAG

→ Format to create tag name is

docker tag <imageName> <username>/<repoName>:<tagname>

$ docker tag springboot-dockerapp javabyraghu/mytestrepo:latest

STEP#9 PUSH IMAGE INTO DOCKER HUB

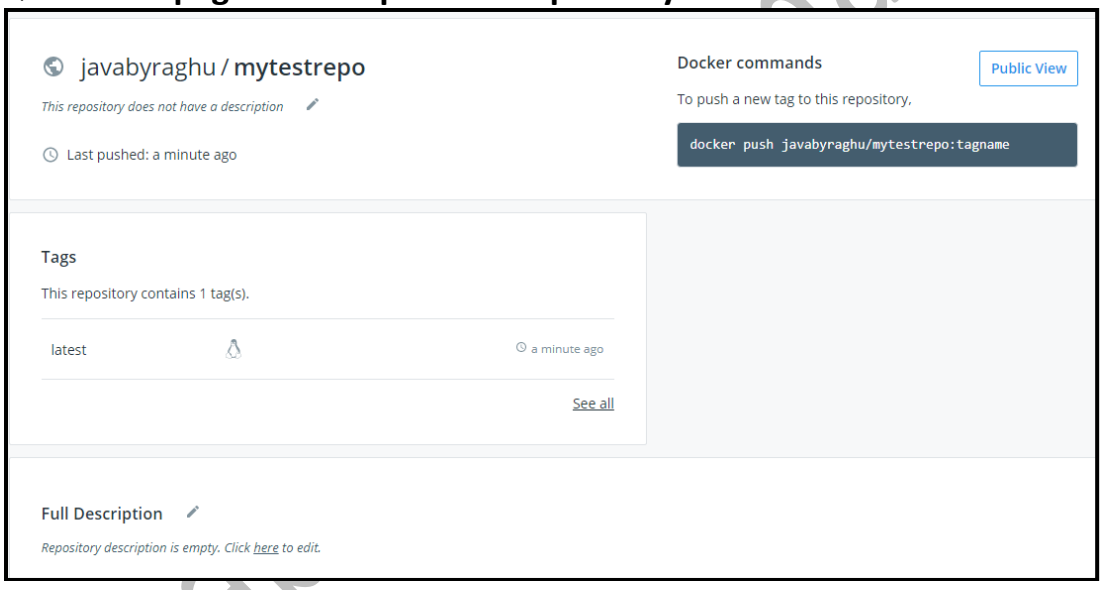
$ docker push <username>/<repoName>:<tagname>

$ docker push javabyraghu/mytestrepo:latest

```
RAGHAVENDRA@RAGHAVENDRA-PC MINGW64 /c/Program Files/Docker Toolbox
$ docker push javabyraghu/mytestrepo:latest
The push refers to repository [docker.io/javabyraghu/mytestrepo]
fd3ee20e03af: Mounted from javabyraghu/onerepo
3a707cf7bc28: Mounted from javabyraghu/onerepo
1690af51cb08: Mounted from javabyraghu/onerepo
5a30999619d7: Mounted from javabyraghu/onerepo
2e669e0134f5: Mounted from javabyraghu/onerepo
8bacec4e3446: Mounted from javabyraghu/onerepo
26b1991f37bd: Mounted from javabyraghu/onerepo
55e6b89812f3: Mounted from javabyraghu/onerepo
latest: digest: sha256:4a140760a8471e1b67b0feee03767065a36732a7ff464b14ae315f5c
05dc851 size: 2007
```

STEP#10 CONFIRM PUSH AT DOCKERHUB

→ Refresh page to see update at repository



FB: https://www.facebook.com/groups/thejavatemple/

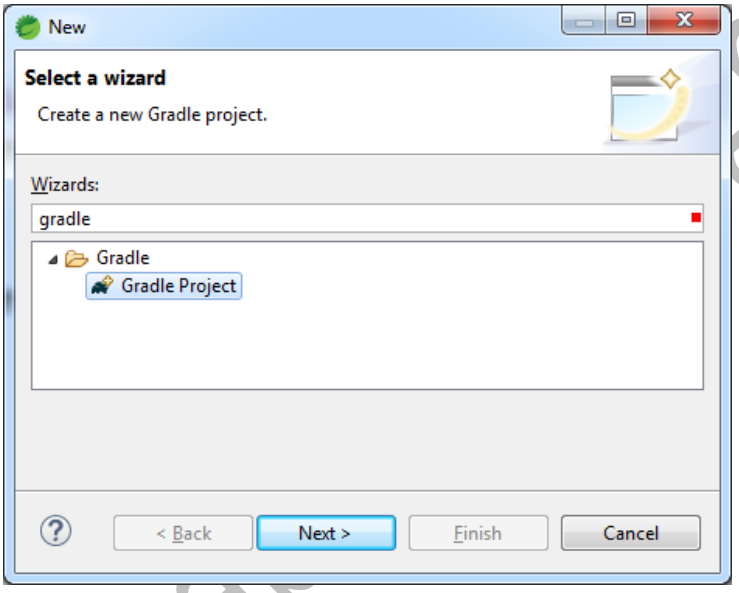
GRADLE

Gradle is general purpose build tool which is similar to Maven but advanced. Gradle API is designed using Groovy language. This is betterment of an internal DSL over XML. Its main purpose is Java projects dependency management and plug-ins.

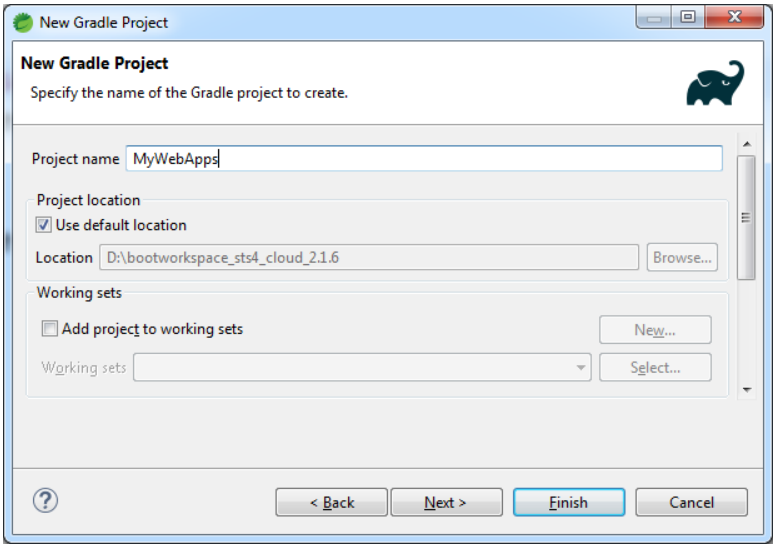
Gradle Servlets/JSP Web Application

1. Create Gradle Project

>File>New>Other...



Enter the name of the project: MyWebApp



> Next > Finish

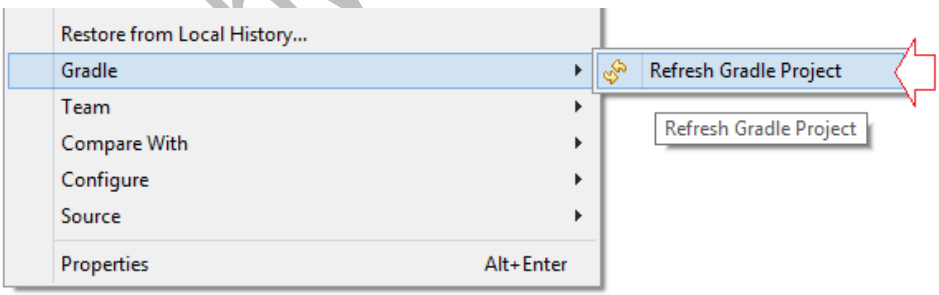
2. Configuring Gradle:-

You need to add the configuration for your application to become "WEB Application". And can be run directly on Eclipse + Tomcat Plugin.

```
apply plugin: 'java'
apply plugin: 'war'
apply plugin: 'com.bmuschko.tomcat'
apply plugin: 'eclipse-wtp'

repositories {
   jcenter()
}

sourceCompatibility = 1.8
targetCompatibility = 1.8

dependencies {
   compile 'com.google.guava:guava:23.0'
   testCompile 'junit:junit:4.12'
   providedCompile "javax.servlet:javax.servlet-api:3.1.0"

}
```

```
dependencies {
   def tomcatVersion = '8.0.53'

   tomcat "org.apache.tomcat.embed:tomcat-embed-core:${tomcatVersion}",
         "org.apache.tomcat.embed:tomcat-embed-logging-
juli:${tomcatVersion}",
         "org.apache.tomcat.embed:tomcat-embed-jasper:${tomcatVersion}"
}


buildscript {

   repositories {
      jcenter()
   }

   dependencies {
      classpath 'com.bmuschko:gradle-tomcat-plugin:2.5'

   }
}
```

Note that each time there is a change in build.gradle you need to update the project, using the tool of Gradle.



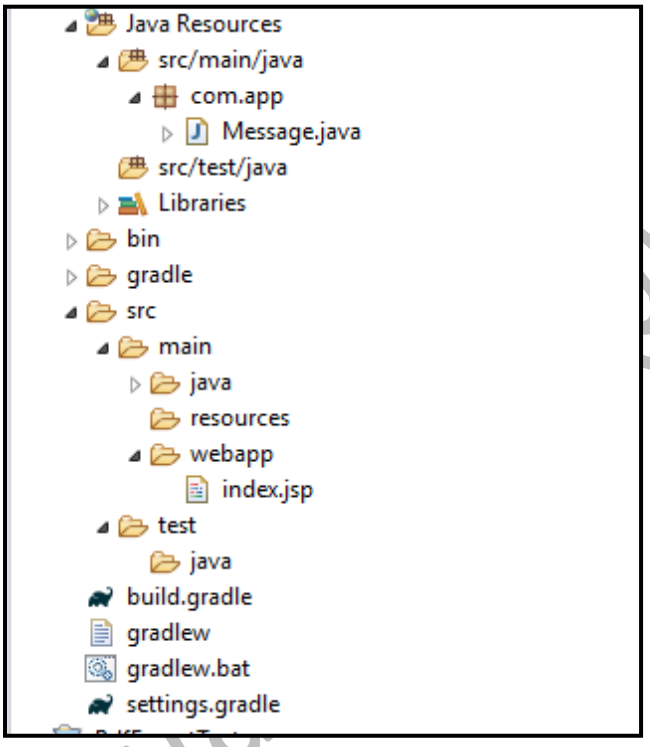
3. Add Folder To Application:-

In "src/main" folder, you need to create 2 sub folders
are "resources" and "webapp".
src/main/java: This folder has java sources.

src/main/resources: This folder can hold property files and other resources

src/main/webapp: This folder holds jsp and other web application content.

4. Write Code in Project



Servlet class:

```
package com.app;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
```

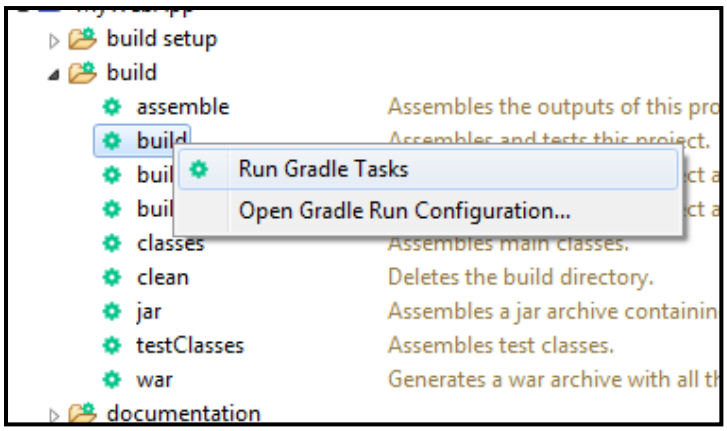
```
import javax.servlet.http.HttpServletResponse;

@WebServlet("/home")
public class Message extends HttpServlet{

	private static final long serialVersionUID = -4633811587840173475L;

	@Override
	protected void doGet(HttpServletRequest req, HttpServletResponse
resp)
			throws ServletException, IOException {
		PrintWriter out=resp.getWriter();
		out.println("Hello App");
	}
}
```

index.jsp file

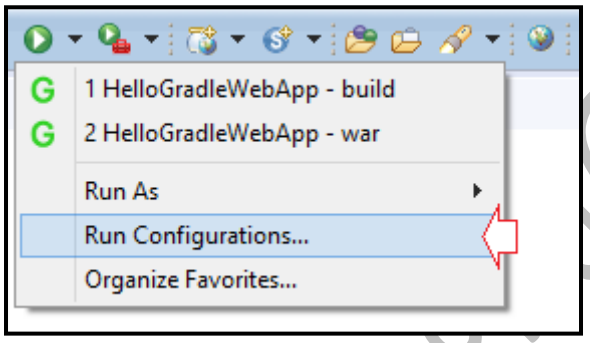
```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<h1>Welcome to All</h1>
</body>
</html>
```


5. Execute Gradle Build Task

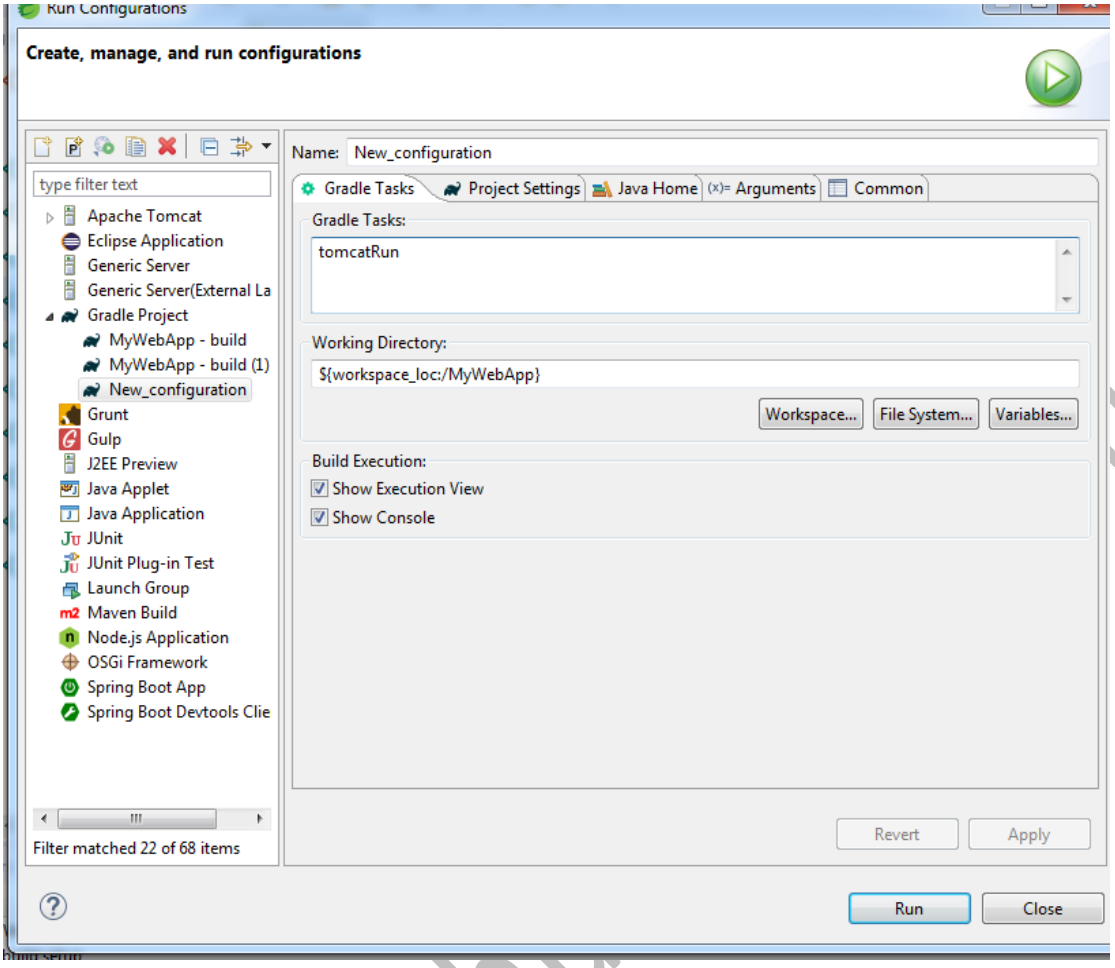
> Open "Gradle Task" view > Expand Project > choose build > Run Gradle Tasks



6. CONFIGURE TO RUN APPLICATION



Enter: tomcatRun > Apply > Run



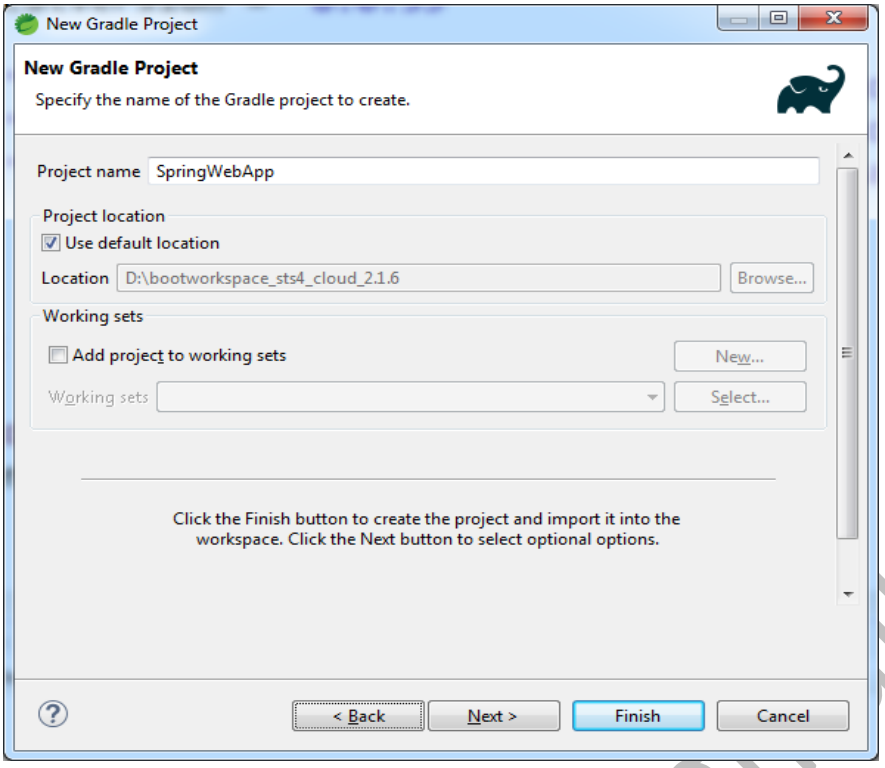
Goto browser and enter URL:

http://localhost:8080/MyWebApp/home

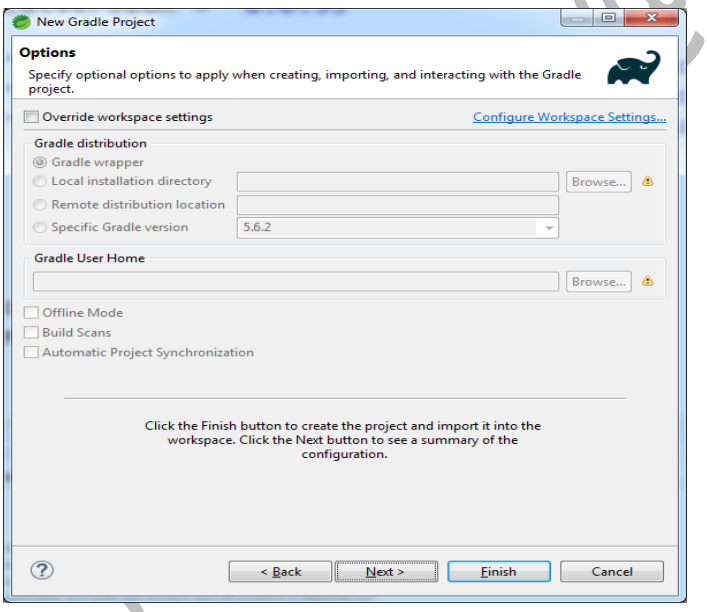
Spring Web Application using Gradle

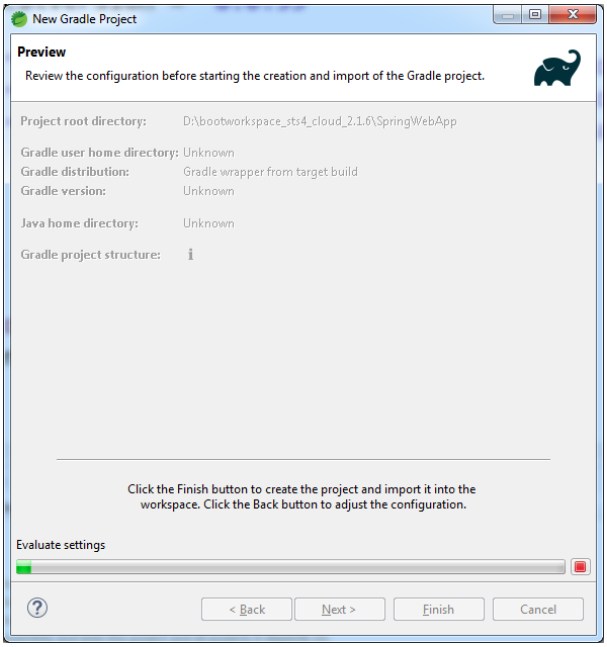
STEP#1 CREATE NEW GRADLE PROJECT

> File > New > Gradle Project > Enter Project Name : SpringWebApp

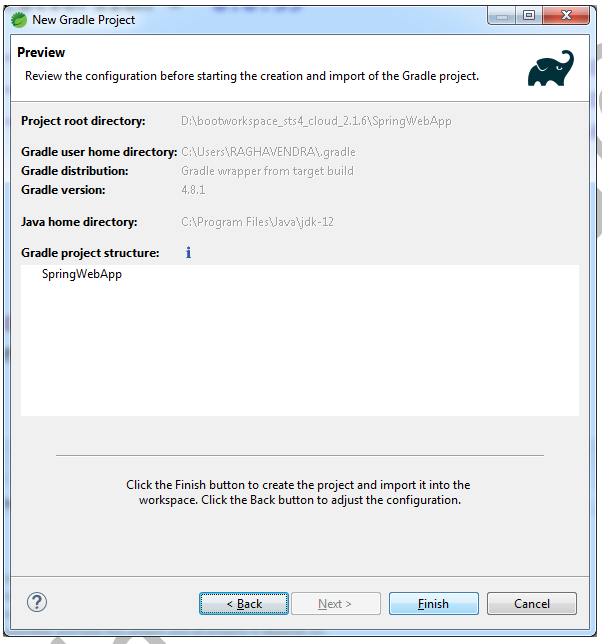


Click on Next Button

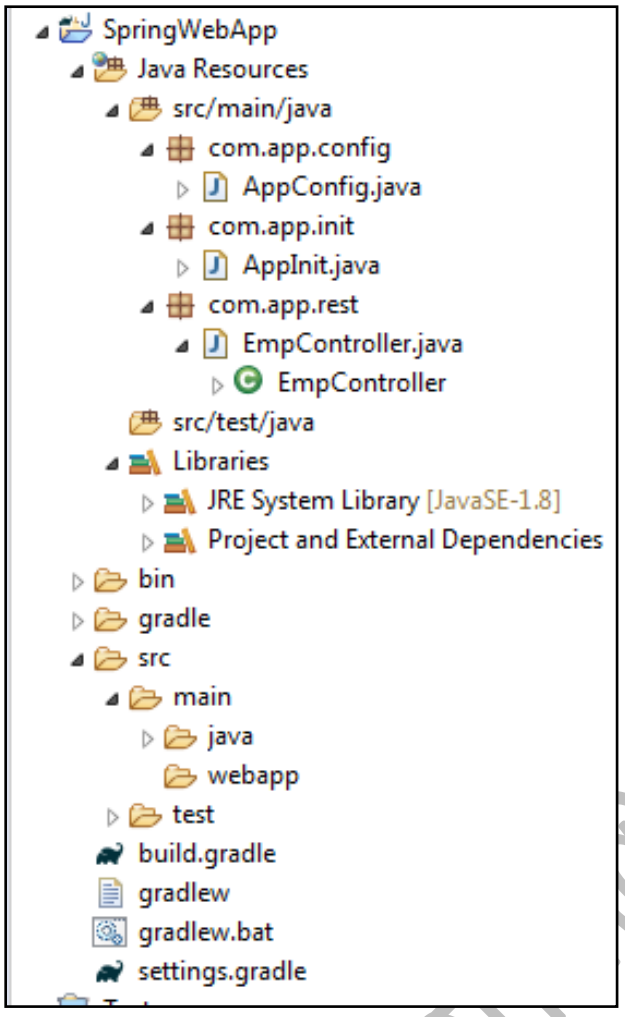




Click on Finish Button



Folder System:-



STEP#2 PROVIDE CODE IN PROJECT

build.gradle

```
apply plugin: 'java'
apply plugin: 'war'
apply plugin: 'com.bmuschko.tomcat'
apply plugin: 'eclipse-wtp'

sourceCompatibility = 1.8
targetCompatibility = 1.8

repositories {
  jcenter()
}

dependencies {
```

```
  providedCompile 'javax.servlet:javax.servlet-api:3.1.0'
  compile 'org.springframework:spring-webmvc:5.1.8.RELEASE'
  compile 'com.fasterxml.jackson.core:jackson-databind:2.9.5'
  runtime 'javax.servlet:jstl:1.2'

  def tomcatVersion = '8.0.53'

   tomcat "org.apache.tomcat.embed:tomcat-embed-core:${tomcatVersion}",
        "org.apache.tomcat.embed:tomcat-embed-logging-
juli:${tomcatVersion}",
        "org.apache.tomcat.embed:tomcat-embed-jasper:${tomcatVersion}"
}
buildscript {

   repositories {
      jcenter()
   }

   dependencies {
      classpath 'com.bmuschko:gradle-tomcat-plugin:2.5'

   }
}
```

AppInit.java

```
package com.app.init;

import
org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcher
ServletInitializer;

import com.app.config.AppConfig;

public class AppInit extends
AbstractAnnotationConfigDispatcherServletInitializer {

        @Override
        protected Class<?>[] getRootConfigClasses() {
```

```
            return null;
        }

        @Override
        protected Class<?>[] getServletConfigClasses() {
            return new Class[] {AppConfig.class};
        }

        @Override
        protected String[] getServletMappings() {
            return new String[] {"/*"};
        }

}
```

AppConfig.java

```
package com.app.config;

import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;

@Configuration
@ComponentScan("com.app")
@EnableWebMvc
public class AppConfig {


}
```

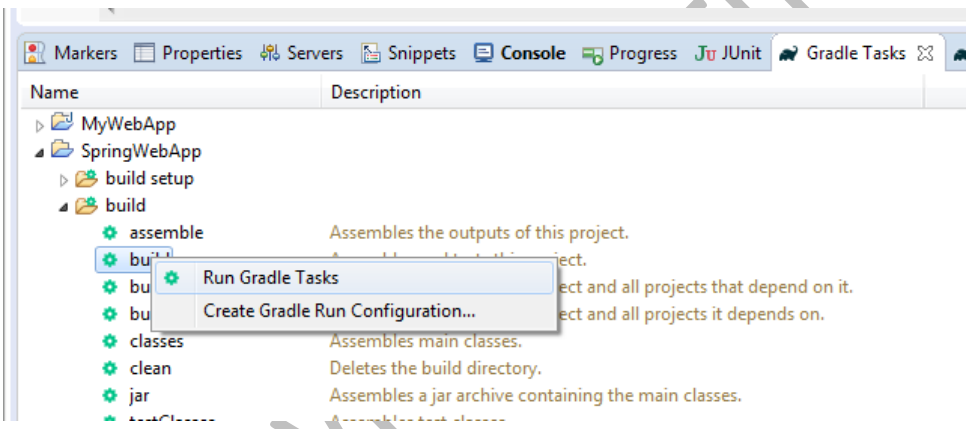
RestController

```
package com.app.rest;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
```

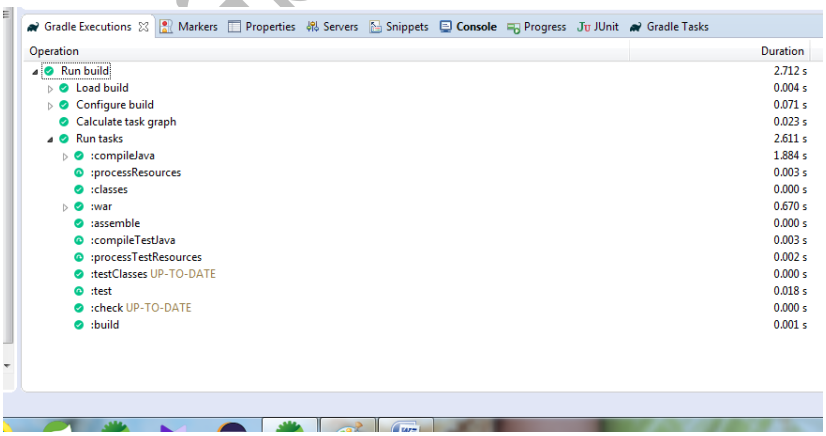
```
@RestController
@RequestMapping("/emp")
public class EmpController {

	@GetMapping("/msg")
	public String show() {
		return "Hello R-APP";
	}
}
```

> Refresh Project

STEP#3 RUN GRADLE TASK

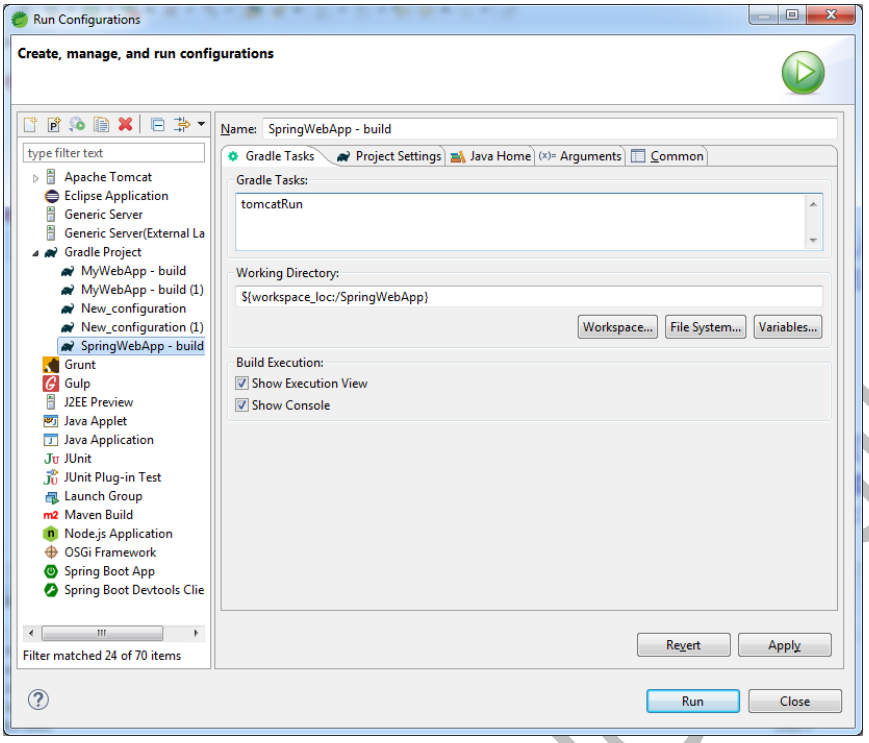


> See Success Message Here



STEP#4 RUN APPLICATION IN TOMCAT

> Right click on Project > Run As > enter tomcatRun > Apply and Run



STEP#5 ENTER URL IN BROWSER

http://localhost:8080/SpringWebApp/emp/msg

email : javabyraghu@gmail.com

FB:

https://www.facebook.com/groups/thejavatemple/