JACOCO
Java Code Coverage

INDEX

JaCoCo - Java Code Coverage

JaCoCo

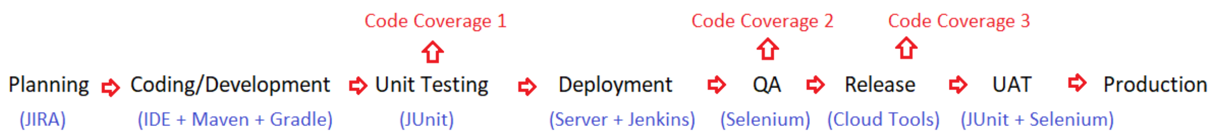
- JaCoCo is Java based Code coverage tool.

Q. What is code coverage?
Ans. It is process of knowing how much percentage of our code is participating in the execution while performing different types automated tests.

Q. Where it will be benefited?
Ans.

a. Dev Team - To check their unit testing is taking place properly or not.
b. QA Team - To check QA/ testers team, automated testing is happening properly or not.
c. Release - To check whether released is properly Tested or not.
d. Client organization - Dev Team Code Coverage + QA Team Code coverage + Release Code coverage helps client organization to be sure about quality of the code.

Where JaCoCo (Code coverage tool) fits in Project life cycle



- At different phase of SDLC the code coverage takes place to just confirm how much percentage our app/ project code is participating in the execution against different tests and test cases we execute.

Different Java based code coverage tools

a. JaCoCo
b. EclEmma
c. Cobetura
d. ncover – For .Net

- SonarQube is basically analysis tool which internally uses JaCoCo for code coverage (SonarQube = JaCoCo++).
- Analysis tool means
  - Checks coding standards followed or not?
  - Checks company rules followed or not?
  - Checks code is vulnerable or not (code is weak or strong for hacking)?

Q. What JaCoCo checks?

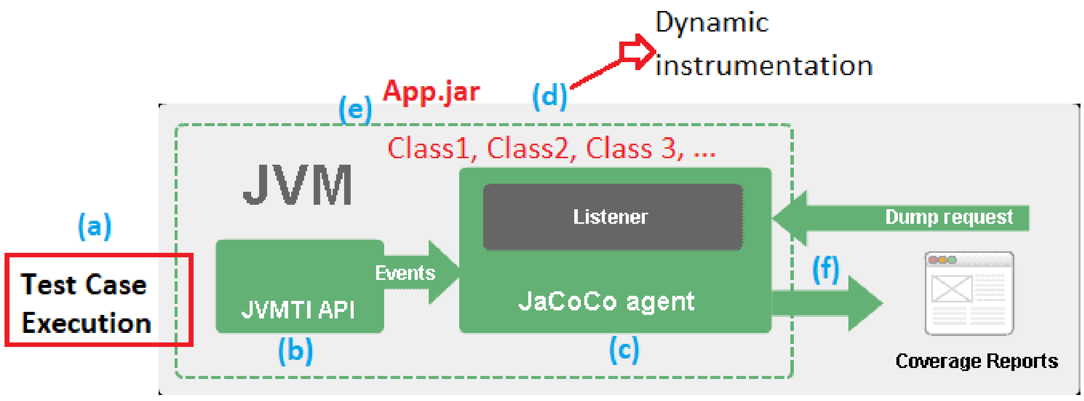
Ans. It checks,

- Line is executed or not
- Method is executed or not
- Branches (if - else condition) is executed or not
- Class is executed or not
- Package is executed or not

Q. How does JaCoCo do this work of code coverage?

Ans. It adds JaCoCo agent/ Java agent to JVM, this agent takes given .class files of the app/ project (JAR/ WAR files) performs dynamic instrumentation (adding extra code to byte code) and performs code coverage activities when the testing events (when Junit, selenium and other automated test cases are executed) are raised on code and also generates the report.

Note: The way Lombok API generates the common code directly to the given class (byte code) based on annotations we add like @Data, @Setter and etc. comes under byte code instrumentation.



Q. Wha is the different Code coverage and Test coverage?

Ans.

- Test coverage speaks about the percentage of test cases that are execute.
- Code coverage speaks about the percentage of code (lines, methods, branches, classes, packages and etc.) that is executed through automated tests.

App1.jar (100 classes) - 50 test cases written covering 100 classes

Case 1: 25 test cases executed covering 50 classes

test coverage - 50% code coverage - 50%

Case 2: 25 test cases executed covering 70 classes
test coverage - 50% code coverage - 70%

Case 3: 50 test cases executed covering 100 classes
test coverage - 100% code coverage -100%

How to install JaCoCo

a. Using manual process (not recommended)
   - Download JaCoCo-<version>.jar file run it - this will keep JaCoCo agent is JVM)

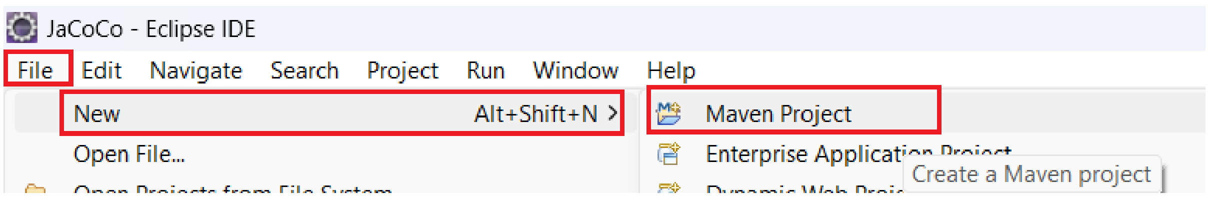
b. Using maven/gradle build tools (Best)
   - Add maven-jacoco plugin to pom.xml (Maven)
   - Add JaCoCo plugin to build.gradel (Gradle)

Q. What is difference b/w maven Surefire plugin and JaCoCo plugin?
Ans.
- Surefire plugin gives info [report about current project/Application listing number of packages, classes, methods, test reports and etc.
- JaCoCo plugin gives code coverage report containing the percentage of code that is executed against automated tests.

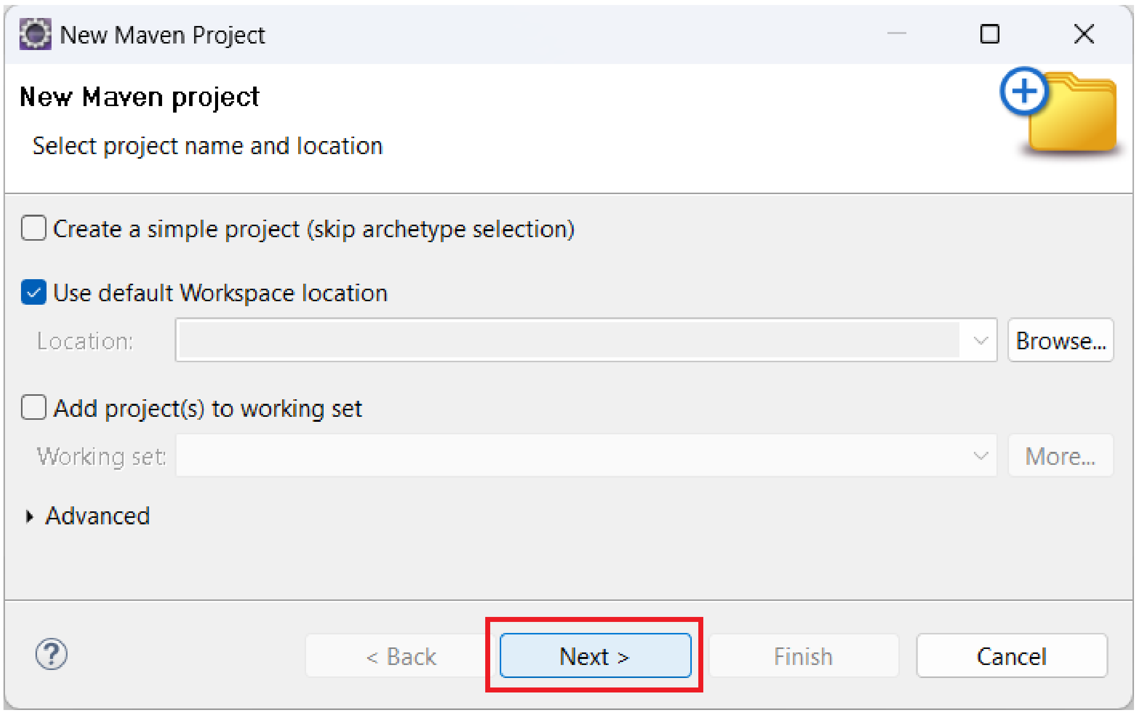
Note:
- ✓ Code coverage does not perform functionality testing/ functionality coverage i.e. it does not tell whether specific functionality is covered or not. Code coverage says whether current testcases of automated testing are sufficient to test our app/project code or not.
- ✓ Recent eclipse versions are giving "EclEmma" as the built-in plugin.

Procedure to develop example app on Code Coverage

Step 1: Create maven Project using "maven-archetype-quickstart". For that click on **File** then click on the **New** option after that you can click on **Maven Project option.**



Step 2: Then one popup will come click on the **Next** button.


New Maven Project

New Maven project

Select project name and location

[ ] Create a simple project (skip archetype selection)

[✓] Use default Workspace location

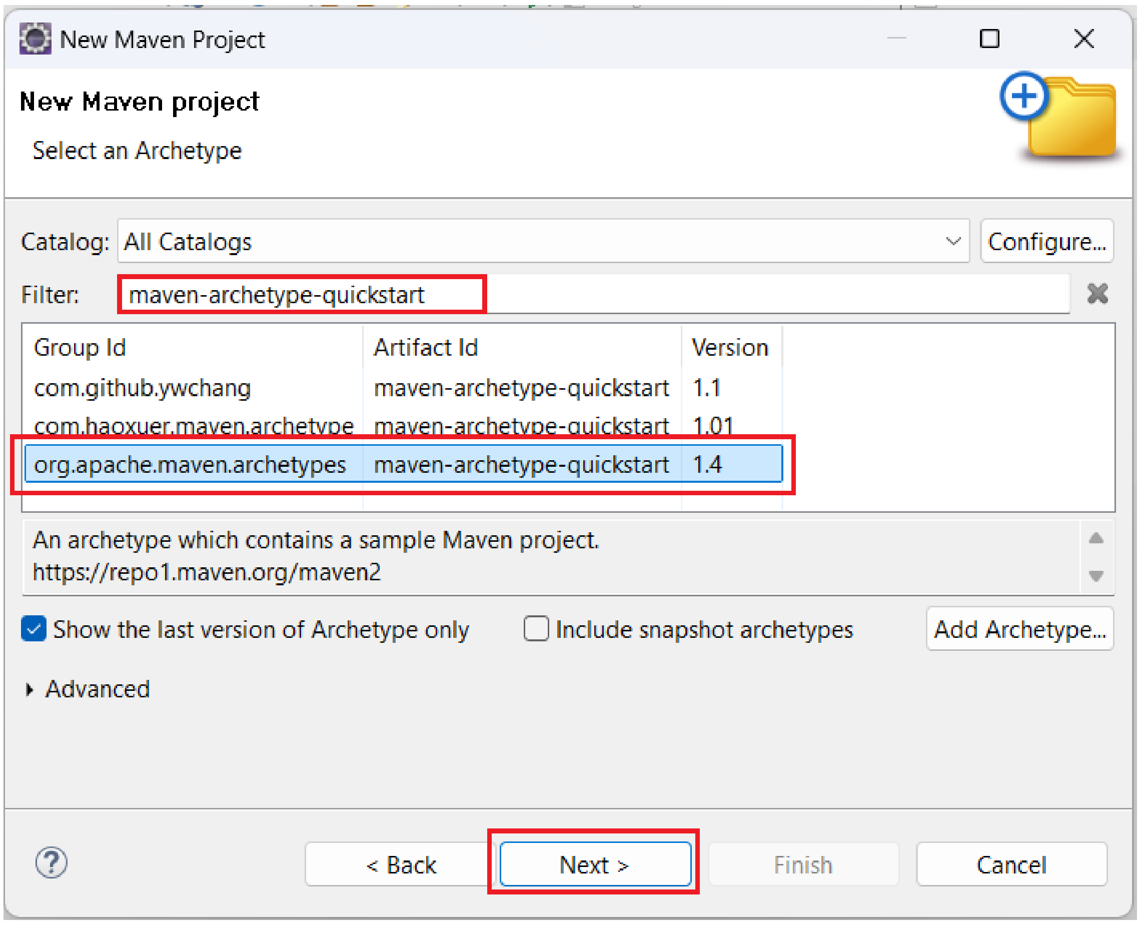
Location:

[ ] Add project(s) to working set

Working set:

Advanced

Step 3: Search **maven-archetype-quickstart** in **Filter** then click on **Next** button.


New Maven Project

New Maven project

Select an Archetype

Catalog: All Catalogs

Filter: maven-archetype-quickstart

Group Id	Artifact Id	Version
com.github.ywchang	maven-archetype-quickstart	1.1
com.haoxuer.maven.archetype	maven-archetype-quickstart	1.01
org.apache.maven.archetypes	maven-archetype-quickstart	1.4

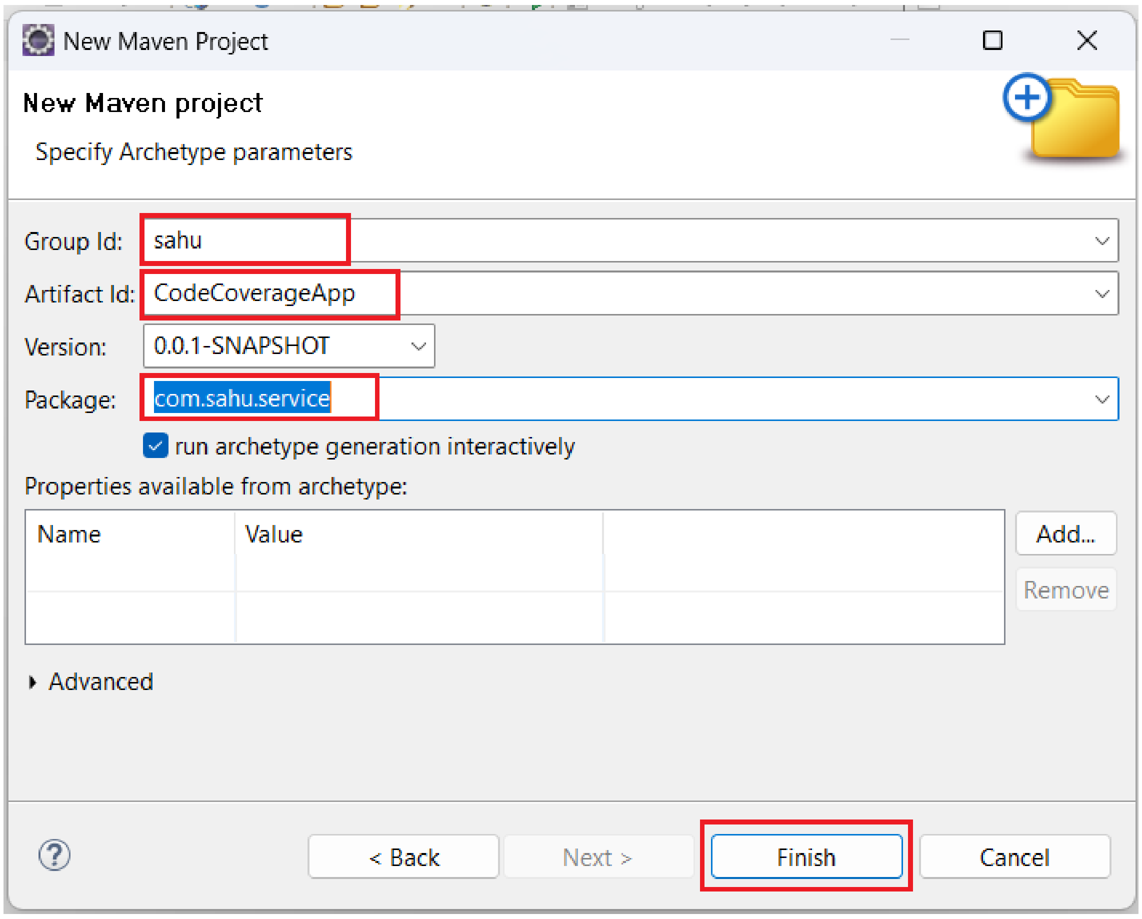
An archetype which contains a sample Maven project.
https://repo1.maven.org/maven2

[✓] Show the last version of Archetype only [ ] Include snapshot archetypes

Advanced

Step 4: Now give the following details as below an click on **Finish** button.

Group Id: sahu
Artifact Id: CodeCoverageApp
Package: com.sahu.service



Group Id: sahu
Artifact Id: CodeCoverageApp
Version: 0.0.1-SNAPSHOT
Package: com.sahu.service
✓ run archetype generation interactively
Properties available from archetype:

Name	Value

Step 5: Change Java version to latest or according to your requirement and do the maven update project.

pom.xml

```
//..............................
<properties>
   <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
   <maven.compiler.source>17</maven.compiler.source>
   <maven.compiler.target>17</maven.compiler.target>
  </properties>
```

Step 6: Add maven-jacoco plugin to pom.xml file [Official Link], [Refined Plugin Link] again we can update.

Note: Use code snippet [GitHub Link].

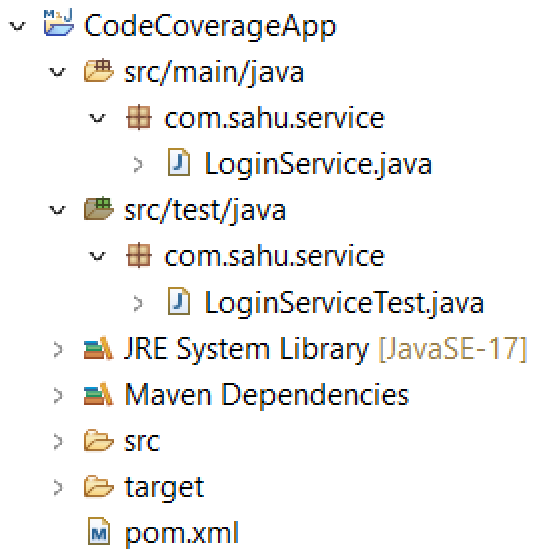
pom.xml

```
      //……………………
<build>
   <pluginManagement><!-- lock down plugins versions to avoid using
Maven defaults (may be moved to parent pom) -->
     <plugins>
                  <!-- JaCoCo Plugin -->
                  <plugin>
                     <groupId>org.jacoco</groupId>
                     <artifactId>jacoco-maven-plugin</artifactId>
                     <version>0.8.2</version>
                     <executions>
                        <execution>
                           <goals>
                              <!-- Adds JaCoCo agent to JVM -->
                              <goal>prepare-agent</goal>
                           </goals>
                        </execution>
                        <execution>
                           <!-- Generates the report during maven "test" phase
(where unit testing code executes automatically) -->
                           <id>report1</id>
                           <phase>test</phase>
                           <goals>
                              <goal>report</goal>
                           </goals>
                        </execution>
                     </executions>
                  </plugin>

      <!— Other plugins …….- ->

     </plugins>
   </pluginManagement>
 </build>
</project>
```

Step 7: Develop the service class having business logic in business method.

Step 8: Write JUnit test cases.

Directory Structure of CodeCoverageApp:



```
CodeCoverageApp
  src/main/java
    com.sahu.service
      LoginService.java
  src/test/java
    com.sahu.service
      LoginServiceTest.java
  JRE System Library [JavaSE-17]
  Maven Dependencies
  src
  target
  pom.xml
```

- Develop the above directory structure using maven archetype quickstart option.
- Create all the package and java classes.
- Then place the following code with in their respective files.

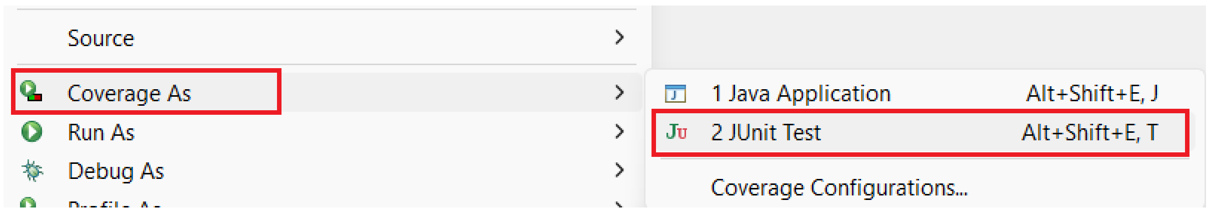
LoginService.java

```
package com.sahu.service;

public class LoginService {

	public String login(String userName, String password) {
		if ((userName == null || userName.length() == 0) || (password
== null || password.length() == 0)) {
			throw new IllegalArgumentException("Invalid inputs");
		}

		if(userName.equalsIgnoreCase("Raja") &&
password.equalsIgnoreCase("Rani")) {
			return "Valid credentials";
		}

		return "Invalid credentials";
	}

}
```

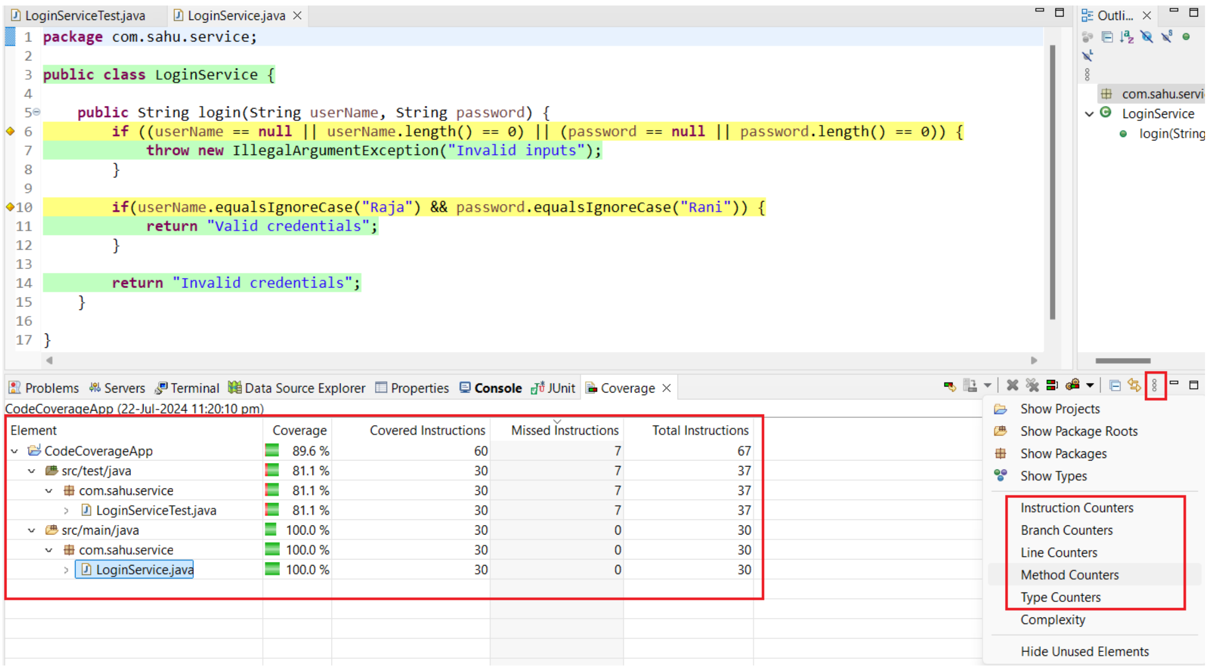
LoginServiceTest.java

```java
package com.sahu.service;

import static org.junit.Assert.assertEquals;

import org.junit.Test;

public class LoginServiceTest {

	private LoginService loginService = new LoginService();

	@Test
	public void testWithValidCredentials() {
		String result = loginService.login("Raja", "Rani");
		assertEquals("Test 1: To check for valid credentials", "Valid 
credentials", result);
	}

	@Test
	public void testWithInValidCredentials() {
		String result = loginService.login("Raja", "Rani1");
		assertEquals("Test 2: To check for Invalid credentials", "Invalid 
credentials", result);
	}

	@Test(expected = IllegalArgumentException.class)
	public void testWithNoCredentials() {
		loginService.login("", "");
	}

}
```

Step 9: To perform code coverage right click on project click on **Coverage As** then click on **JUnit Test**.


Source
Coverage As
Run As
Debug As
1 Java Application Alt+Shift+E, J
2 JUnit Test Alt+Shift+E, T
Coverage Configurations...

Step 10: See the Reports, by click on right side 3 dot you can get so many option and we can see all the report just explore little bit with all the options.



```java
package com.sahu.service;

public class LoginService {

    public String login(String userName, String password) {
        if ((userName == null || userName.length() == 0) || (password == null || password.length() == 0)) {
            throw new IllegalArgumentException("Invalid inputs");
        }

        if(userName.equalsIgnoreCase("Raja") && password.equalsIgnoreCase("Rani")) {
            return "Valid credentials";
        }

        return "Invalid credentials";
    }

}
```

CodeCoverageApp (22-Jul-2024 11:20:10 pm)

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
CodeCoverageApp	89.6 %	60	7	67
src/test/java	81.1 %	30	7	37
com.sahu.service	81.1 %	30	7	37
LoginServiceTest.java	81.1 %	30	7	37
src/main/java	100.0 %	30	0	30
com.sahu.service	100.0 %	30	0	30
LoginService.java	100.0 %	30	0	30

Note: We need to add more and more test case until service classes code coverage becomes 100% (at least 80%).

--- The END ---