# SWE 434
# Software Testing and Validation

## Code Coverage Concept

*Lecture slides are available on following website through your university student login*
https://lms.ksu.edu.sa/

**Raja Majid Mehmood**
rmehmood@ksu.edu.sa
Department of Software Engineering,
King Saud University, Riyadh, Saudi Arabia.

**Some of the material in these slides is derived from https://confluence.atlassian.com/, Thanks to them and also special thanks to Dr. Shamim**

# Lecture's Agenda

1. Code Coverage
2. Code Coverage Types
3. Computation of Code Coverage

# What is Code Coverage?

- Code coverage is the percentage of code which is covered by automated tests.
- Code coverage measurement simply determines
  - which statements in a body of code have been executed through a test run, and
  - which statements have not.
- Code coverage system collects information about the running program and generate the program analysis report
- Code coverage is part of a feedback loop in the software development process.
  - As tests cases are developed, code coverage highlights aspects of the code
    - which may not be effectively tested and
    - which require additional testing.
  - This loop will continue until coverage meets some specified target.

# Why Measure Code Coverage?

- We measure code coverage for the following reasons:
  - To know how well our tests actually test our code
  - To know whether we have enough testing in place
  - To maintain the test quality over the lifecycle of a project

# Common Types of Code Coverage

| Element Type | Description |
|---|---|
| **Instructions** | ❑ This coverage information is obtained from Java byte code instructions of given program.<br>❑ Instruction coverage provides information about the amount of code that has been executed or missed. |
| **Branches** | It calculates branch coverage for all if and switch statements. This metric counts the total number of branches in a method and determines the number of executed or missed branches. |
| **Cyclomatic Complexity** | It also calculates cyclomatic complexity for each non-abstract method and summarizes complexity for classes, packages and groups. |
| **Statements/ Lines** | For all class files that have been compiled with debug information, coverage information for individual lines can be calculated. |
| **Methods** | Each non-abstract method contains at least one instruction. A method is considered as executed when at least one instruction has been executed. |
| **Classes** | A class is considered as executed when at least one of its methods has been executed. |

# Code Coverage Measurement

- Challenge
  - How to compute the code coverage elements?

# Code Coverage Analysis

## Application Class (Class Under Test)

```
public class MyMath
{
1.  public int div(int a, int b) {

2.  if (b==0)
3.      return -1;

4.      else {
5.          int result = a / b;
6.          return result;
7.      }
    }
}
```

-Method Coverage **MC**:
-line # 1 to be counted

-Branch Coverage **BC:**
-line # 2 or 4 to be counted as ONE

-Statement Coverage **SC:**
- Line # 2 is also counted as a if-else statement
-if (b==0) then line # 3 to be counted
-if (b≠0) then line # 5 & 6 to be counted

# Computing Coverage of *MyMath*

**CASE-1: Assume;**

a = 10, and b= 5 ;
used in **MyMath.*div*** (10, 5)

```
public class MyMath
{
1.  public int div(int a, int b) {

2.      if (b==0)
3.          return -1;
4.      else {
5.          int result = a / b;
6.          return result;
7.      }
    }
}
```

-Method Coverage **MC**:
-line # 1 to be counted

-Branch Coverage **BC:**
-line # 4 to be counted as ONE for else-part

-Statement Coverage **SC:**
- Line # 2 is also counted as a if-else statement
-if (b≠0) then line # 5 & 6 to be counted

```
Method Coverage:      1       (line # 1)
Branch Coverage:      1       (line # 4)
Statements Coverage:  3       (line # 2, 5, 6)
```

# Computing Coverage of *MyMath*

**CASE-2: Assume;**

a = 10, and b= 0 ;

used in **MyMath.*div*** (10, 0)

```
public class MyMath
{
1.  public int div(int a, int b) {

2.      if (b==0)
3.          return -1;
4.      else {
5.          int result = a / b;
6.          return result;
7.      }
    }
}
```

-Method Coverage **MC**:
-line # 1 to be counted

-Branch Coverage **BC**:
-line # 2 to be counted as ONE for true-part

-Statement Coverage **SC**:
- Line # 2 is also counted as a if-else statement
-if (b==0) then line # 3 to be counted

```
Method Coverage:      1       (line # 1)
Branch Coverage:      1       (line # 2)
Statements Coverage:  2       (line # 2, 3)
```

# SWE 434
# Software Testing and Validation

## Clover Installation in Eclipse

*Lecture slides are available on following website through your university student login*
https://lms.ksu.edu.sa/

**Raja Majid Mehmood**
rmehmood@ksu.edu.sa
Department of Software Engineering,
King Saud University, Riyadh, Saudi Arabia.

**Some of the material in these slides is derived from https://confluence.atlassian.com/, Thanks to them and also special thanks to Dr. Shamim**

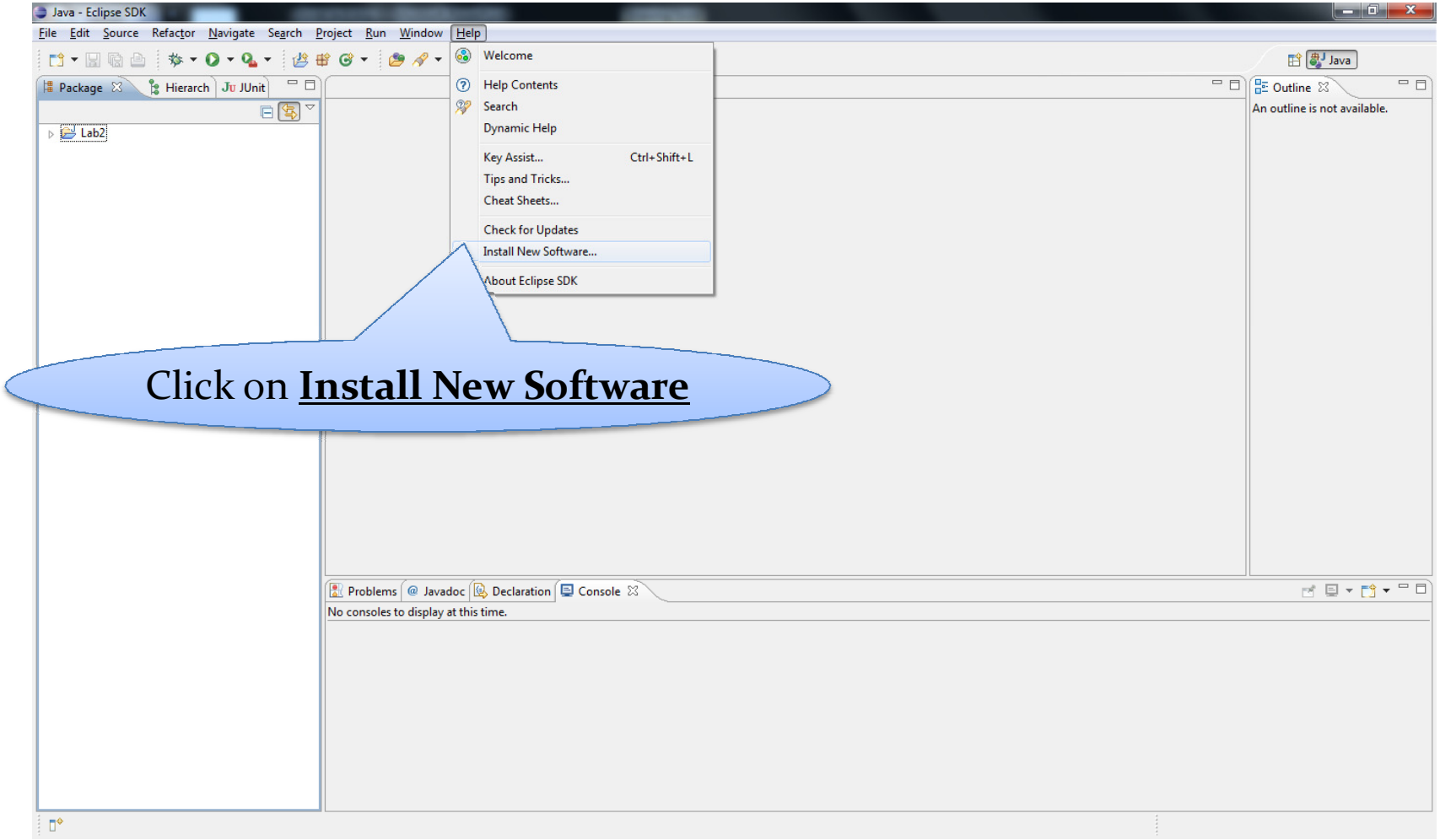
# Lecture's Agenda

1. Clover **3.1.10** Installation Wizard

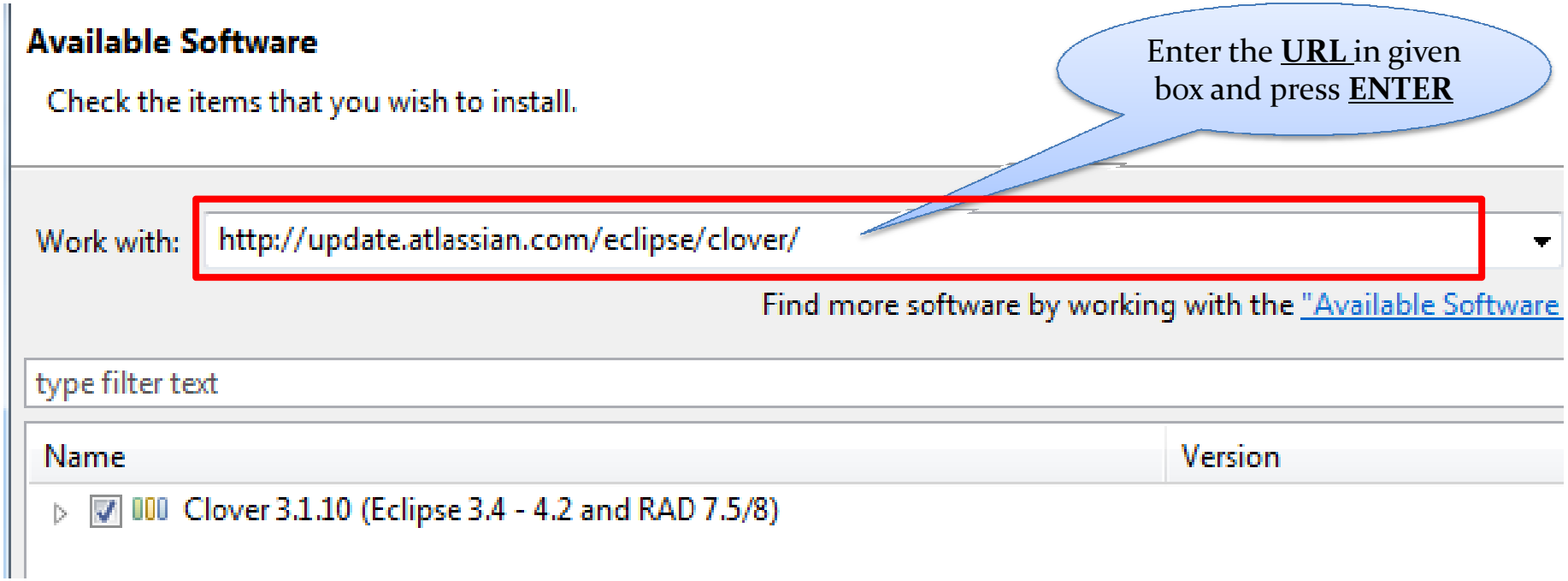
# Clover Installation in Eclipse

Please follow all the steps for successful installation on your personal computers

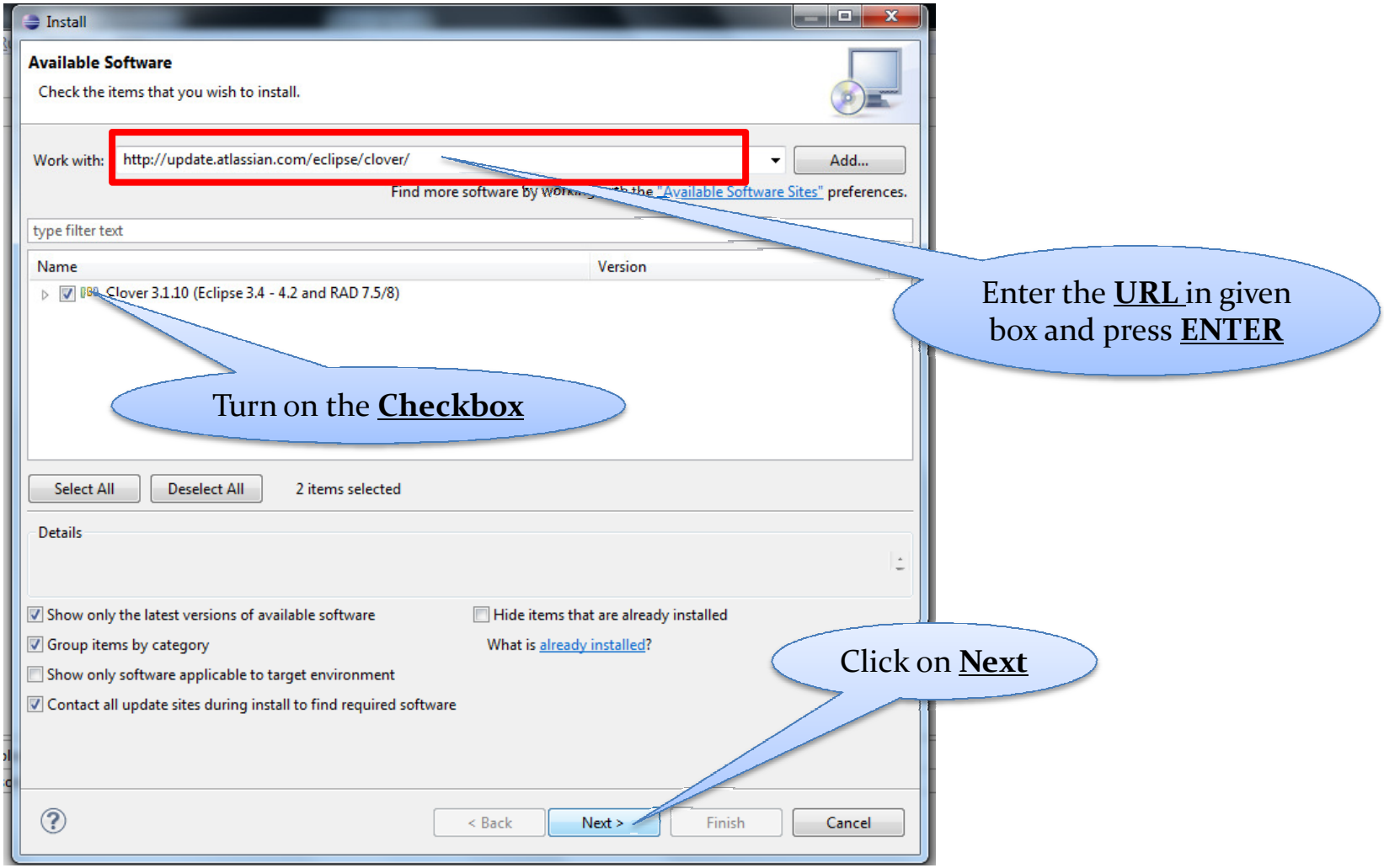
# Clover Installation in Eclipse (-1)



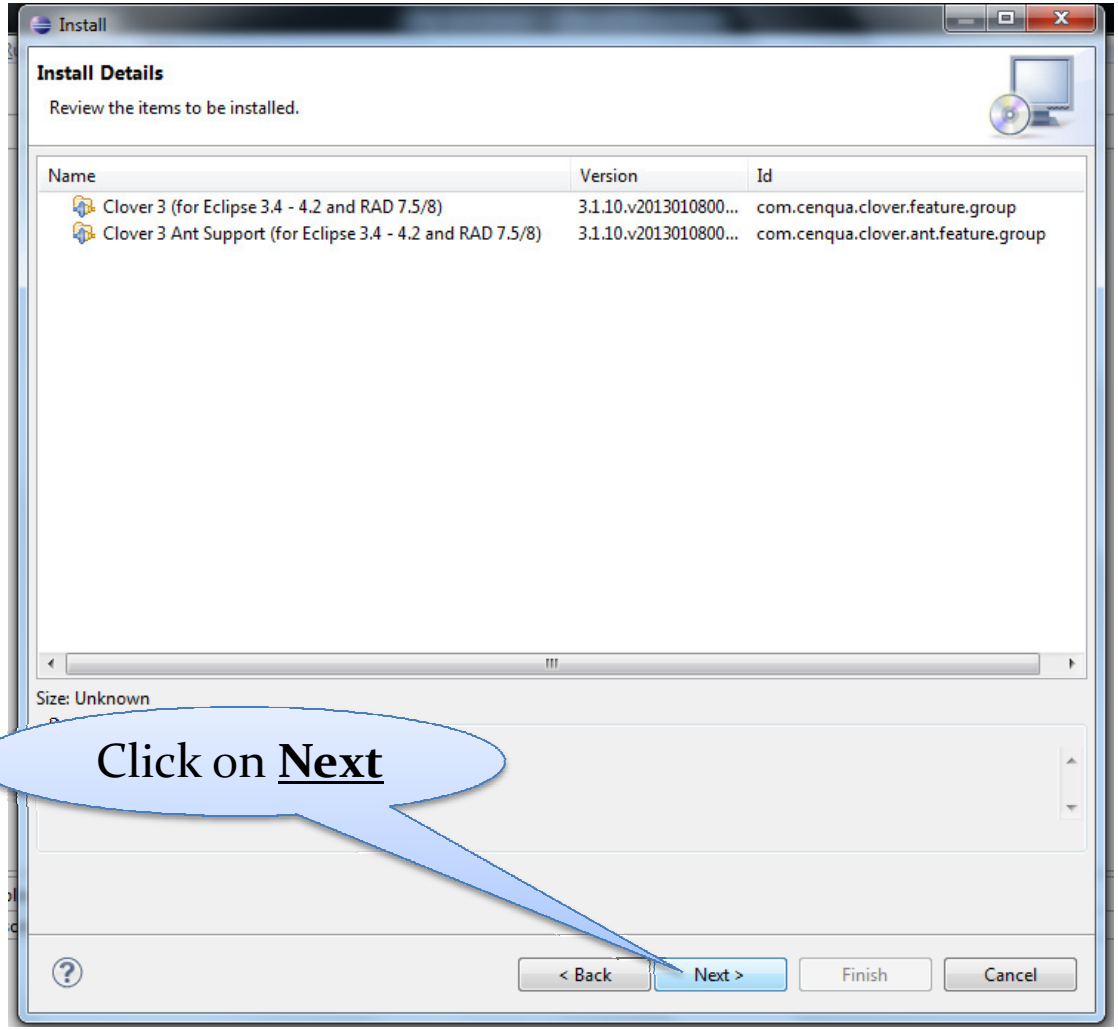
# Step-2


Available Software
Check the items that you wish to install.
Enter the URL in given box and press ENTER
Work with:
http://update.atlassian.com/eclipse/clover/
Find more software by working with the "Available Software
type filter text
Name
Version
Clover 3.1.10 (Eclipse 3.4 - 4.2 and RAD 7.5/8)

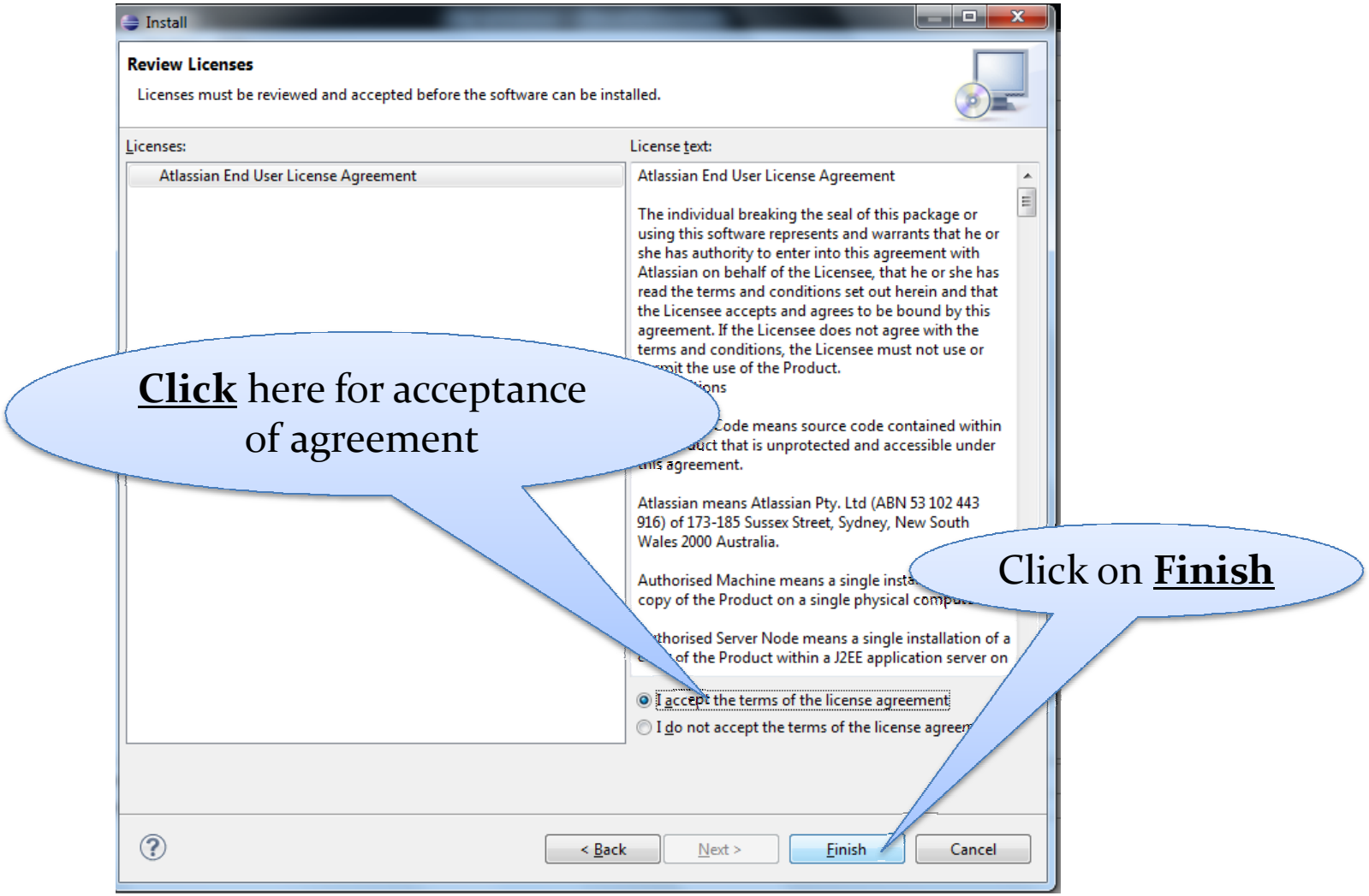
# Step-2


Install
Available Software
Check the items that you wish to install.
Work with:
http://update.atlassian.com/eclipse/clover/
Add...
Find more software by working with the "Available Software Sites" preferences.
type filter text
Name
Version
Clover 3.1.10 (Eclipse 3.4 - 4.2 and RAD 7.5/8)
Select All
Deselect All
2 items selected
Details
Show only the latest versions of available software
Hide items that are already installed
Group items by category
What is already installed?
Show only software applicable to target environment
Contact all update sites during install to find required software
< Back
Next >
Finish
Cancel
Enter the URL in given box and press ENTER
Turn on the Checkbox
Click on Next

# Step-3

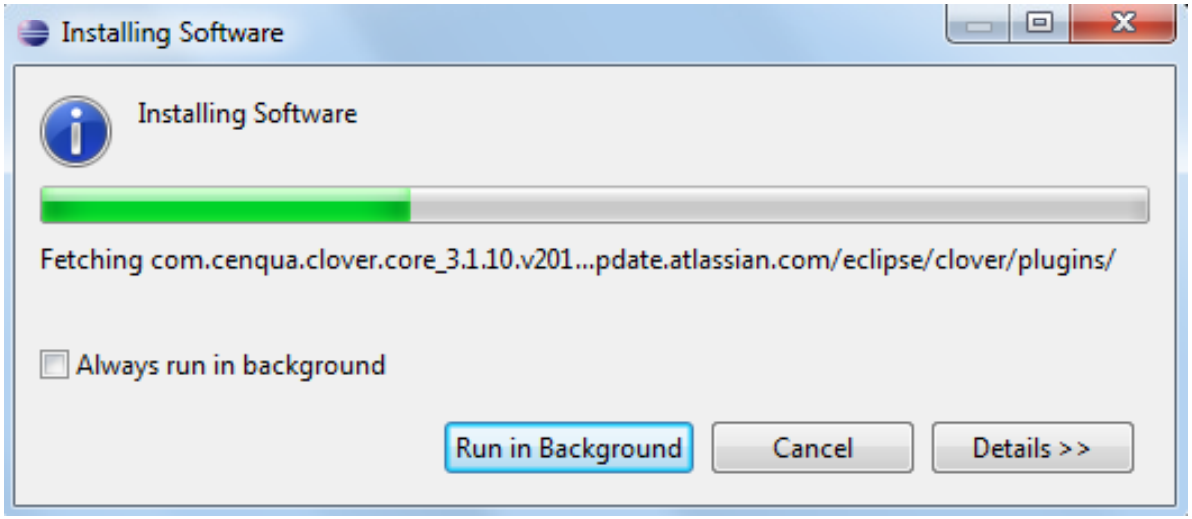

Install
Install Details
Review the items to be installed.
Name
Version
Id
Clover 3 (for Eclipse 3.4 - 4.2 and RAD 7.5/8)
3.1.10.v2013010800...
com.cenqua.clover.feature.group
Clover 3 Ant Support (for Eclipse 3.4 - 4.2 and RAD 7.5/8)
3.1.10.v2013010800...
com.cenqua.clover.ant.feature.group
Size: Unknown
Click on Next
< Back
Next >
Finish
Cancel

# Step-4


Install
Review Licenses
Licenses must be reviewed and accepted before the software can be installed.
Licenses:
Atlassian End User License Agreement
License text:
Atlassian End User License Agreement
The individual breaking the seal of this package or using this software represents and warrants that he or she has authority to enter into this agreement with Atlassian on behalf of the Licensee, that he or she has read the terms and conditions set out herein and that the Licensee accepts and agrees to be bound by this agreement. If the Licensee does not agree with the terms and conditions, the Licensee must not use or
Atlassian means Atlassian Pty. Ltd (ABN 53 102 443 916) of 173-185 Sussex Street, Sydney, New South Wales 2000 Australia.
I accept the terms of the license agreement
I do not accept the terms of the license agreem
< Back
Next >
Finish
Cancel
Click here for acceptance of agreement
Click on Finish

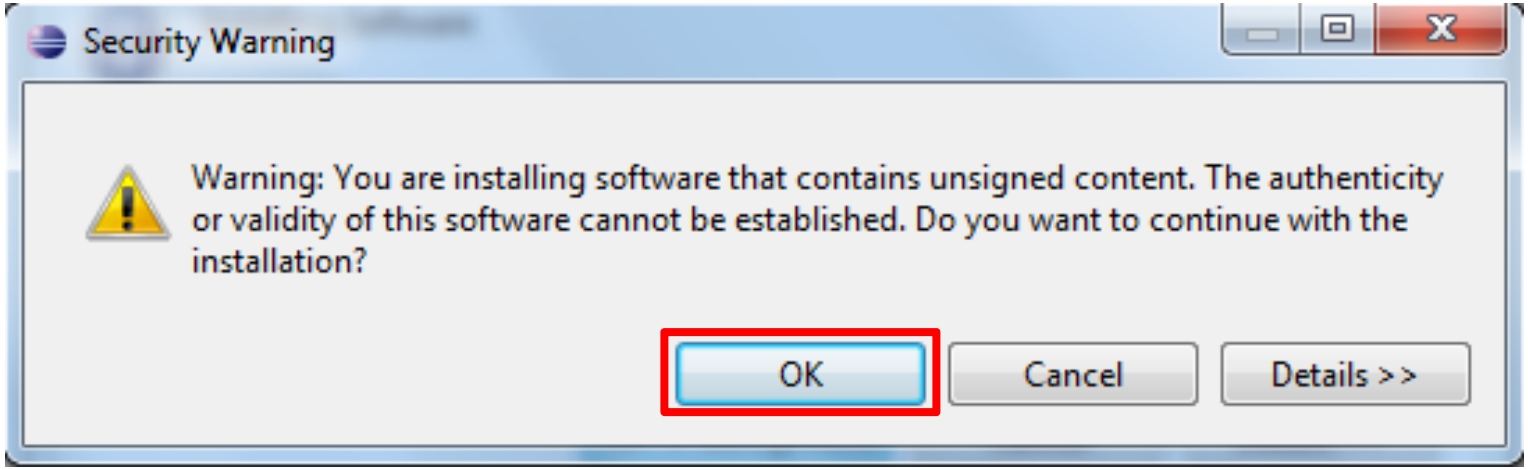
# Step-5

- **Wait** for few minutes
- Also, follow the procedure



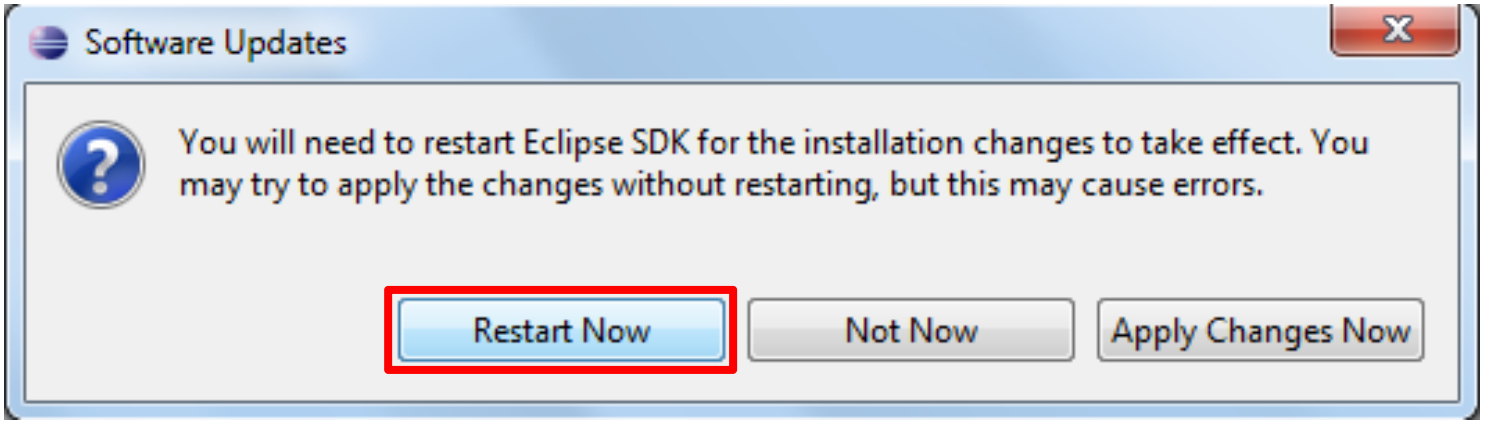
# Step-6

- Click **OK**

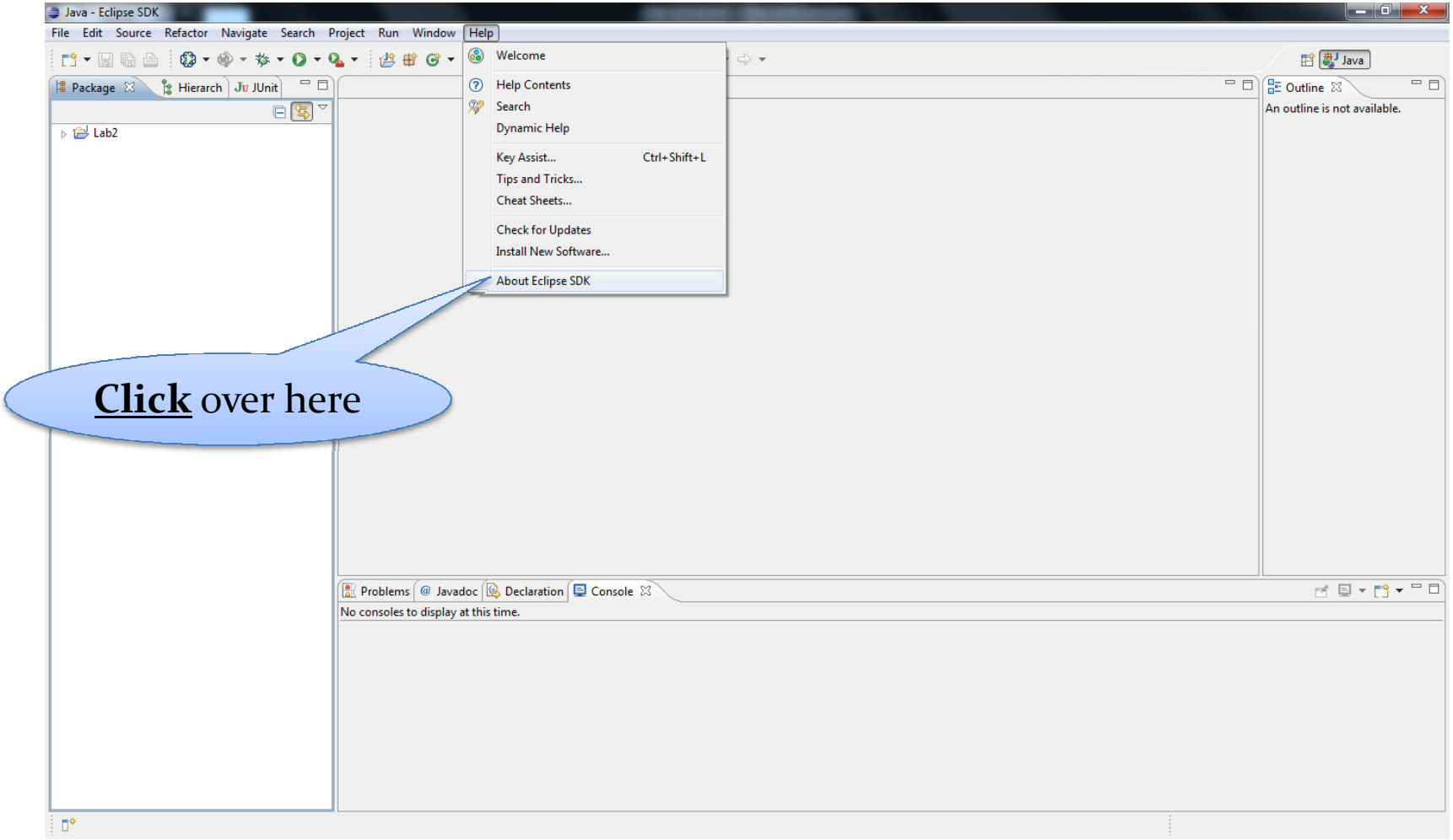


# Step-7

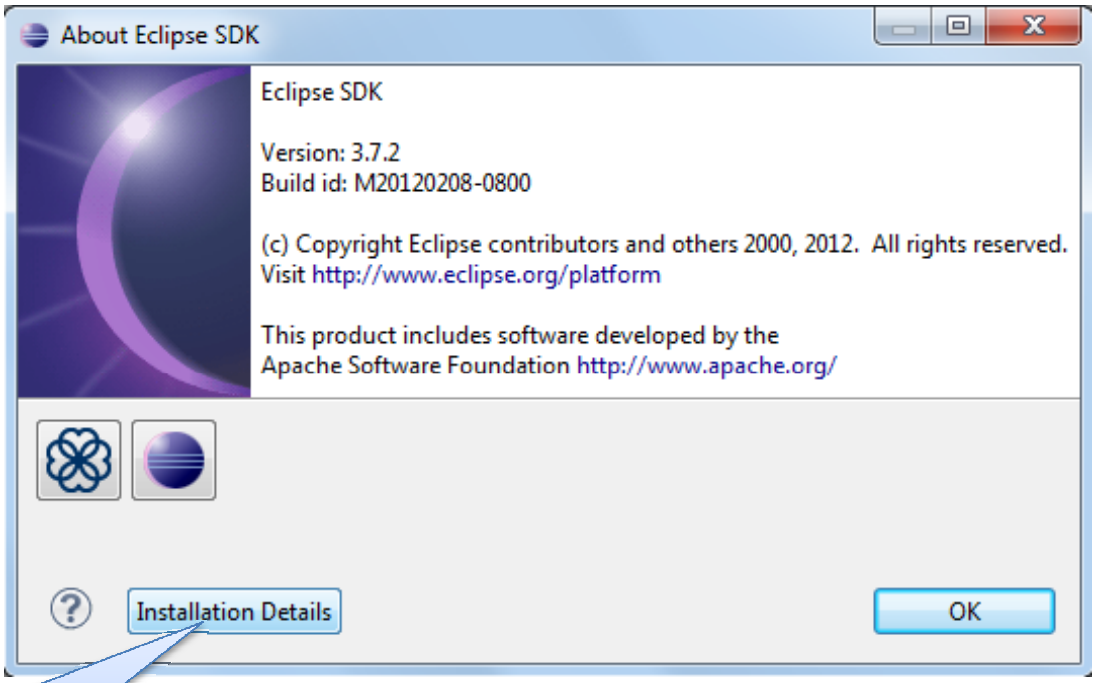
- Click **Restart Now**



# How to confirm Clover in Eclipse (1)

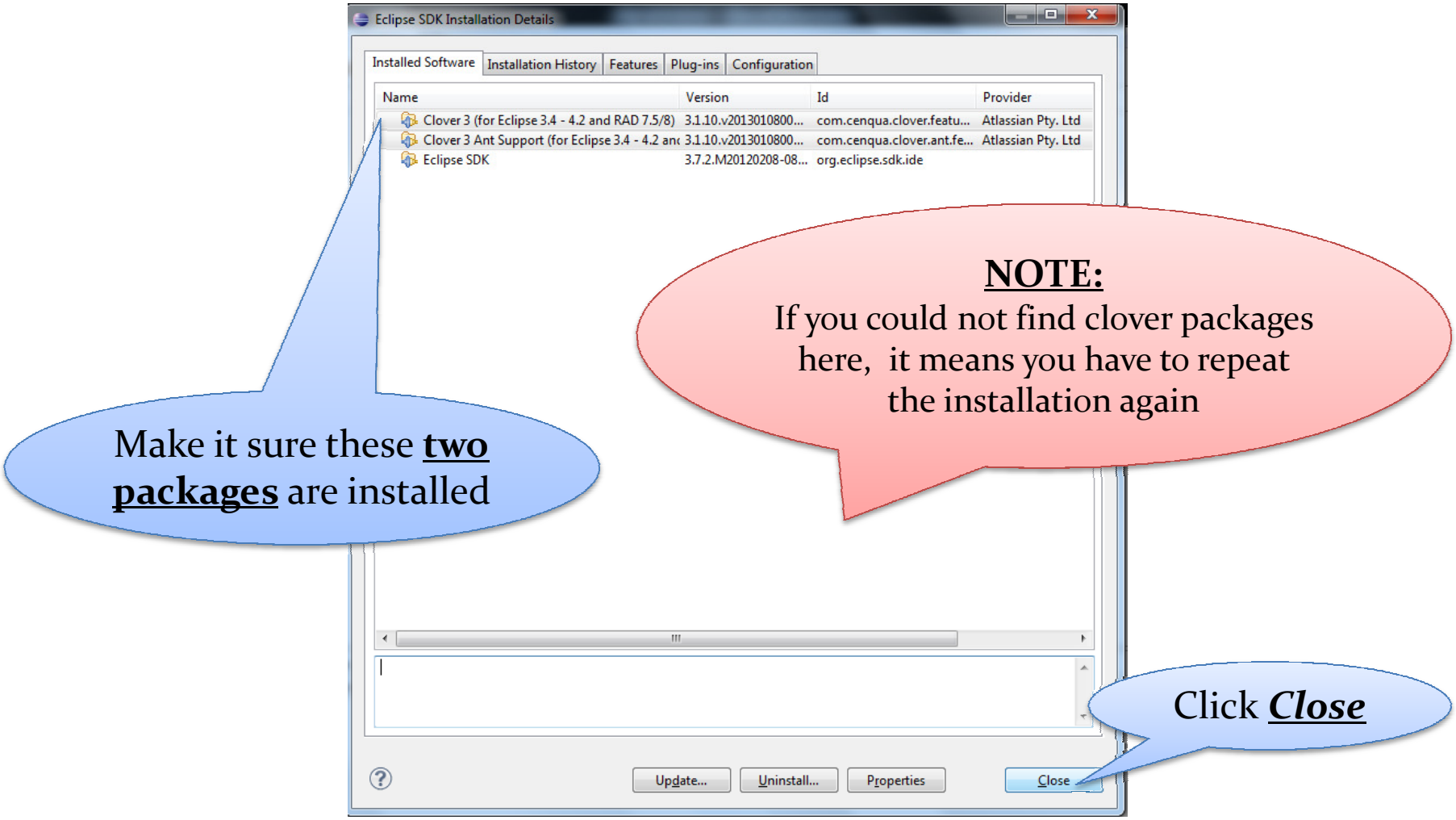


# Step-2

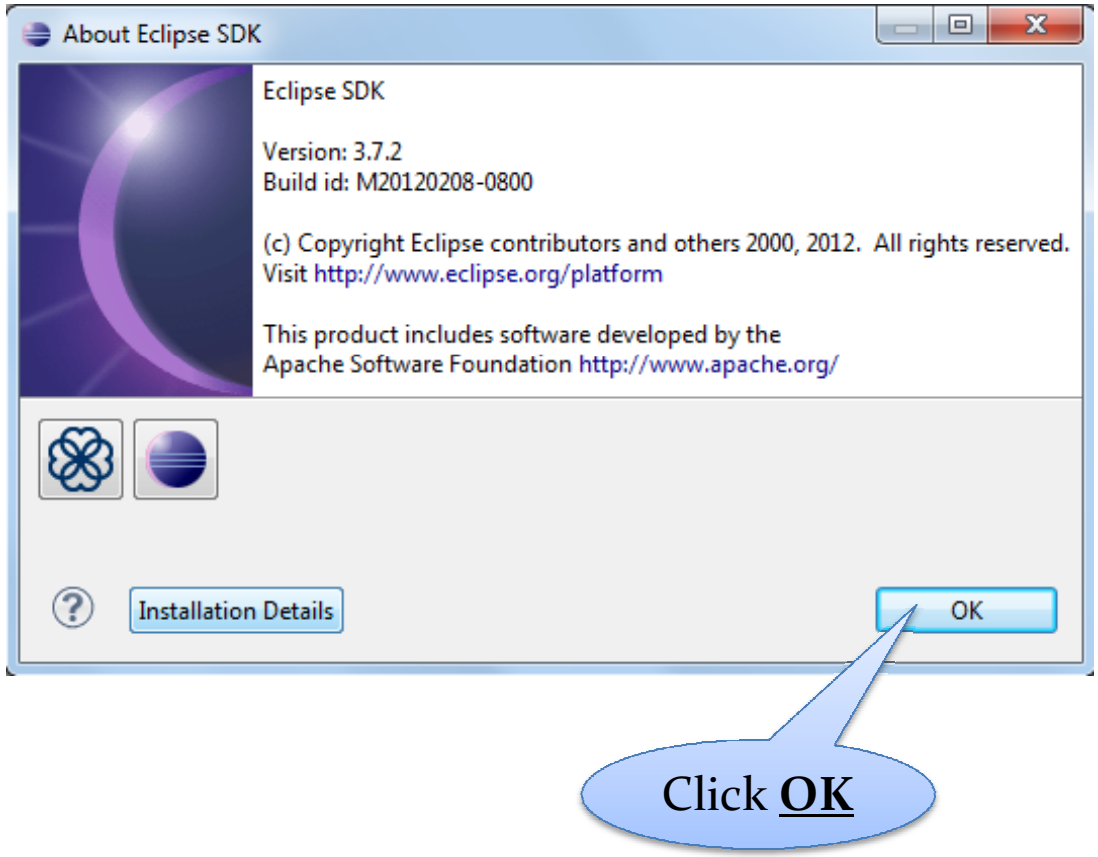



Click on **Installation Details**

# Step-3


Eclipse SDK Installation Details
Installed Software
Installation History
Features
Plug-ins
Configuration
Name
Version
Id
Provider
Clover 3 (for Eclipse 3.4 - 4.2 and RAD 7.5/8)
3.1.10.v2013010800...
com.cenqua.clover.featu...
Atlassian Pty. Ltd
Clover 3 Ant Support (for Eclipse 3.4 - 4.2 an
3.1.10.v2013010800...
com.cenqua.clover.ant.fe...
Atlassian Pty. Ltd
Eclipse SDK
3.7.2.M20120208-08...
org.eclipse.sdk.ide
Update...
Uninstall...
Properties
Close
Make it sure these two packages are installed
NOTE:
If you could not find clover packages here, it means you have to repeat the installation again
Click Close

# Step-4


About Eclipse SDK
Eclipse SDK
Version: 3.7.2
Build id: M20120208-0800
(c) Copyright Eclipse contributors and others 2000, 2012. All rights reserved.
Visit http://www.eclipse.org/platform
This product includes software developed by the
Apache Software Foundation http://www.apache.org/
Installation Details
OK
Click OK

# SWE 434
# Software Testing and Validation

## Practical Example with Clover and JUnit

*Lecture slides are available on following website through your university student login*
https://lms.ksu.edu.sa/

**Raja Majid Mehmood**
rmehmood@ksu.edu.sa
Department of Software Engineering,
King Saud University, Riyadh, Saudi Arabia.

**Some of the material in these slides is derived from https://confluence.atlassian.com/, Thanks to them and also special thanks to Dr. Shamim**

# Lecture's Agenda

1. Main Clover Components
2. How to enable the eclipse project for *Clover*
3. Practical Example (Lab4)
    1. Code Coverage from **0**% to **100**%

# Clover-for-Eclipse User's Guide

- **Visit the below mentioned *URL* for getting more detailed explanation of Clover, and its components.**

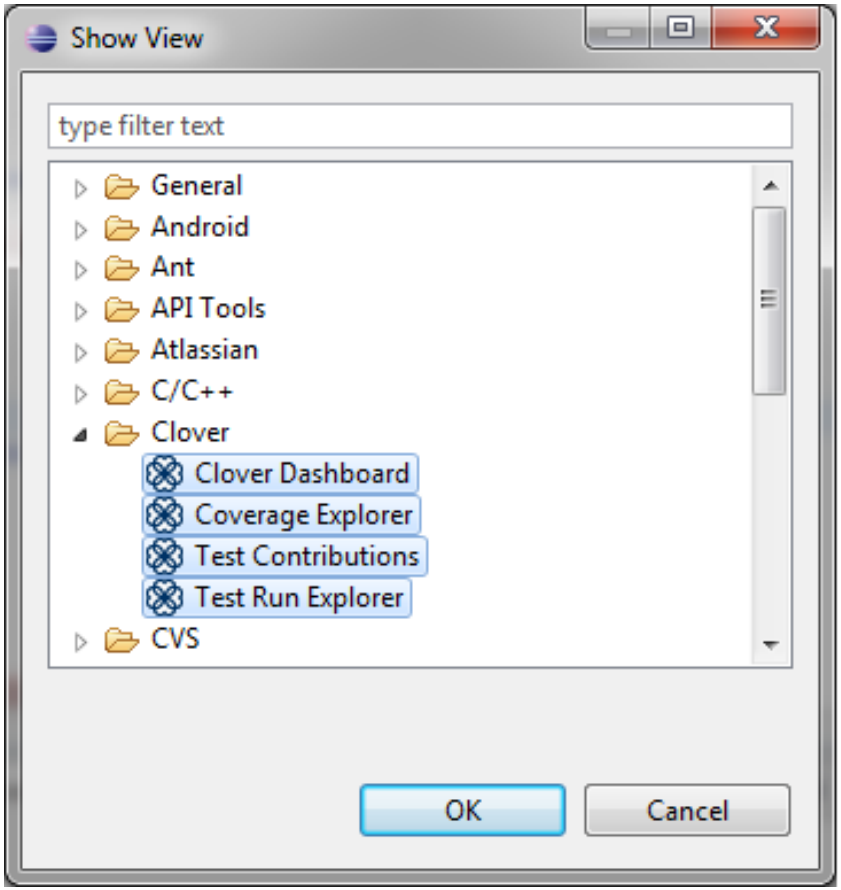
https://confluence.atlassian.com/display/CLOVER/Clover-for-Eclipse+User's+Guide

# Code Coverage with Clover

- Clover is designed to measure code coverage
- Clover's IDE Plugin provide developers with a way to quickly measure code coverage
- Clover's - Eclipse integrations allow coverage measurement to be performed in Automated Environment
- And its reports generated to be shared by the team.

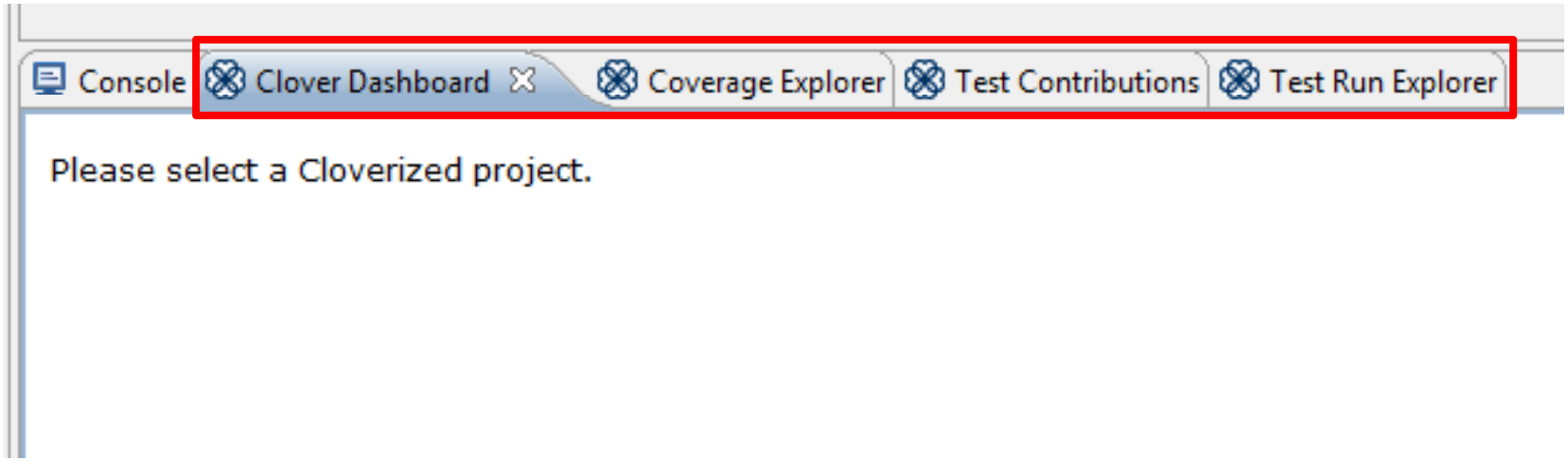
# Clover Components in Eclipse

- You can always open them from
  - *"**Window** > **Show View** > **Other** ... > **Clover**"*
- Four Clover views will be opened automatically:
  - Clover Dashboard
  - Coverage Explorer
  - Test Contributions
  - Test Run Explorer



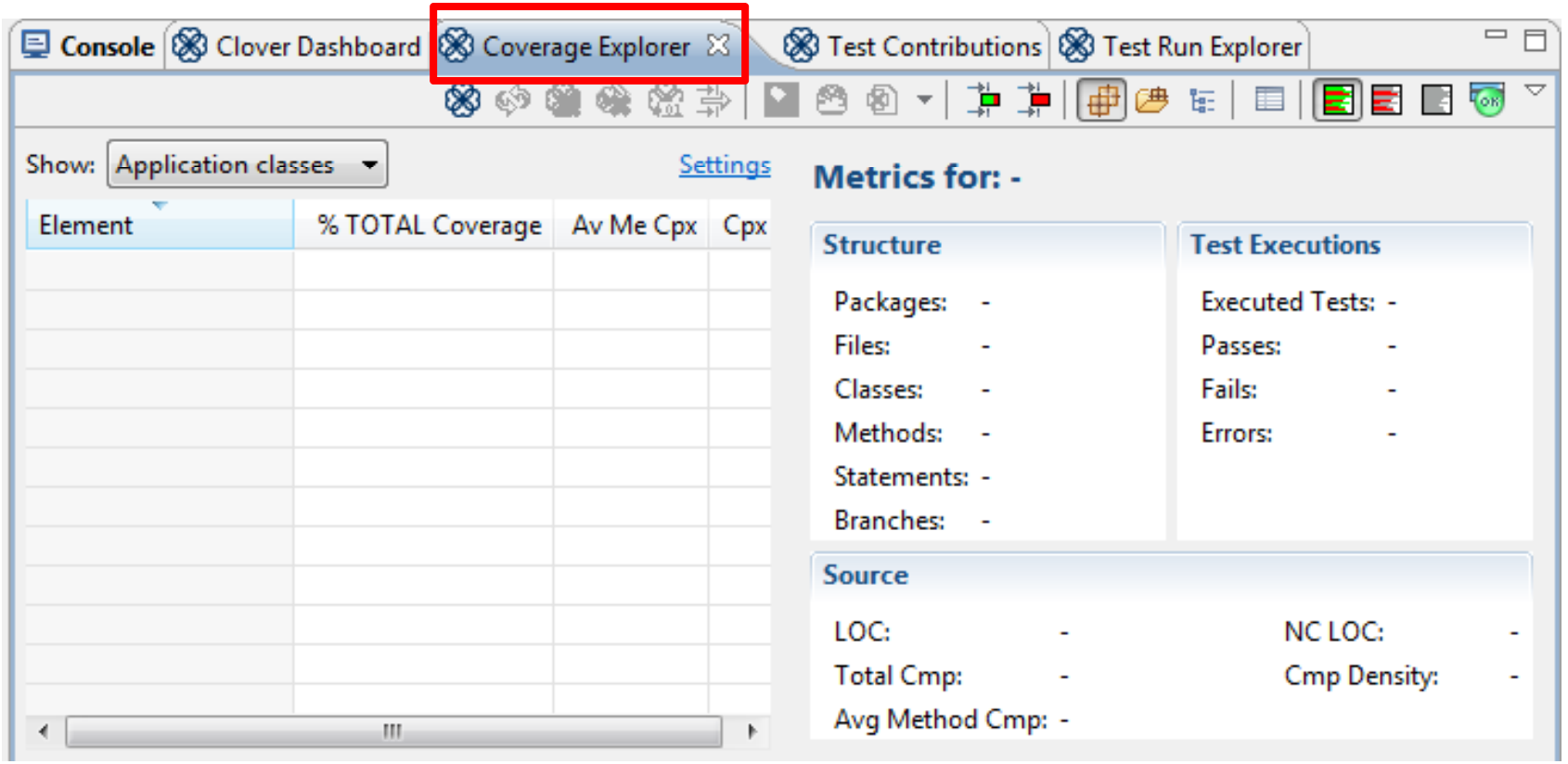
# Clover Components in Eclipse

- We can see this components view in the bottom of Eclipse



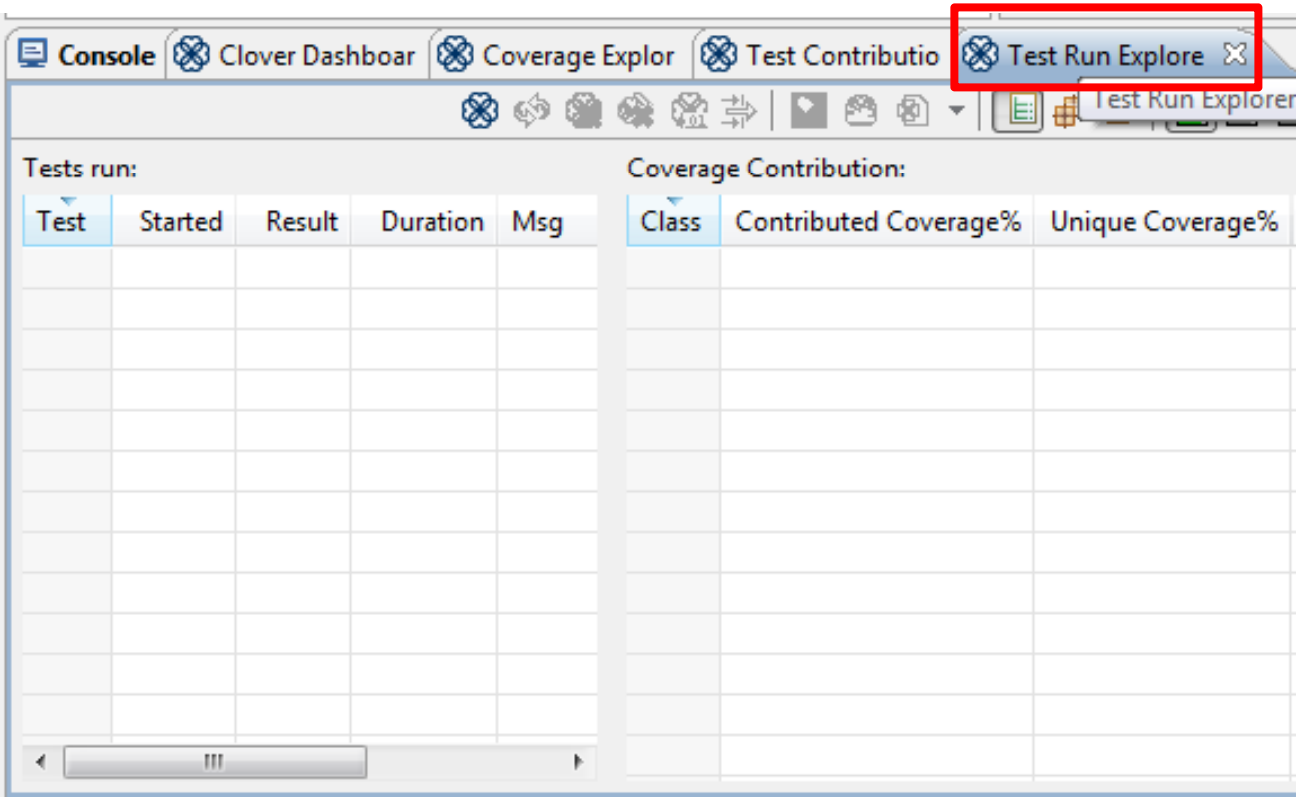
# (1). Coverage Explorer

- It's the best place to start. You will see coverage and metrics for all instrumented projects.



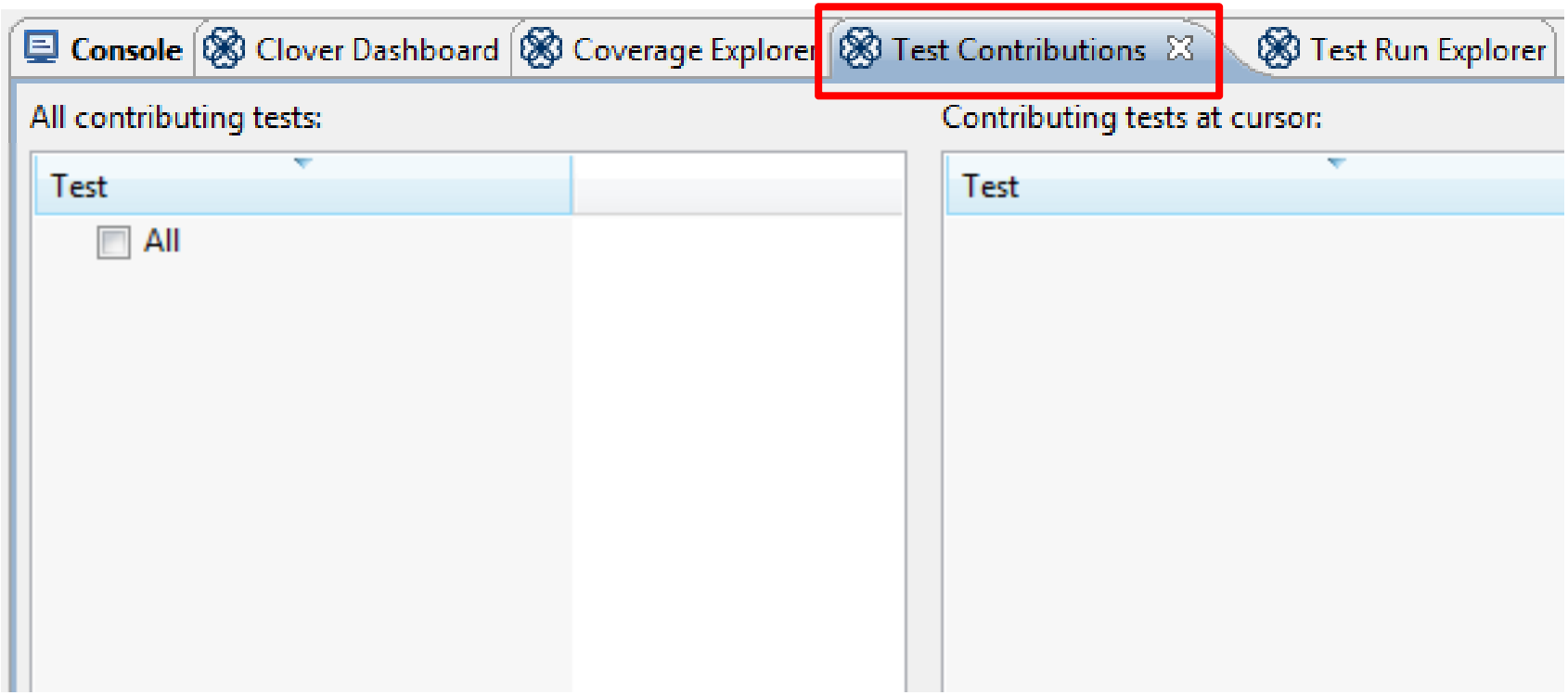
# (2). Test Run Explorer

- The ***Test Run Explorer*** view, lets you explore your recently run tests - showing whether they passed or failed, their duration and any error messages that they generated.



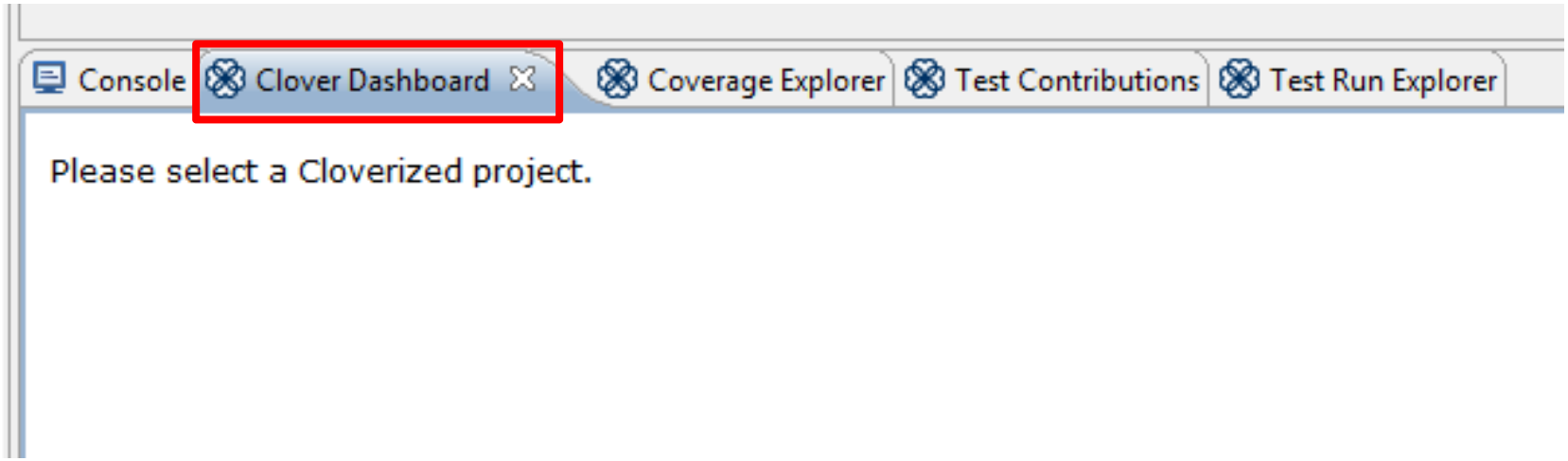
# (3). Test Contributions

- The ***Test Contributions*** view shows unit tests and methods that generated coverage for the currently opened and selected Java source file.

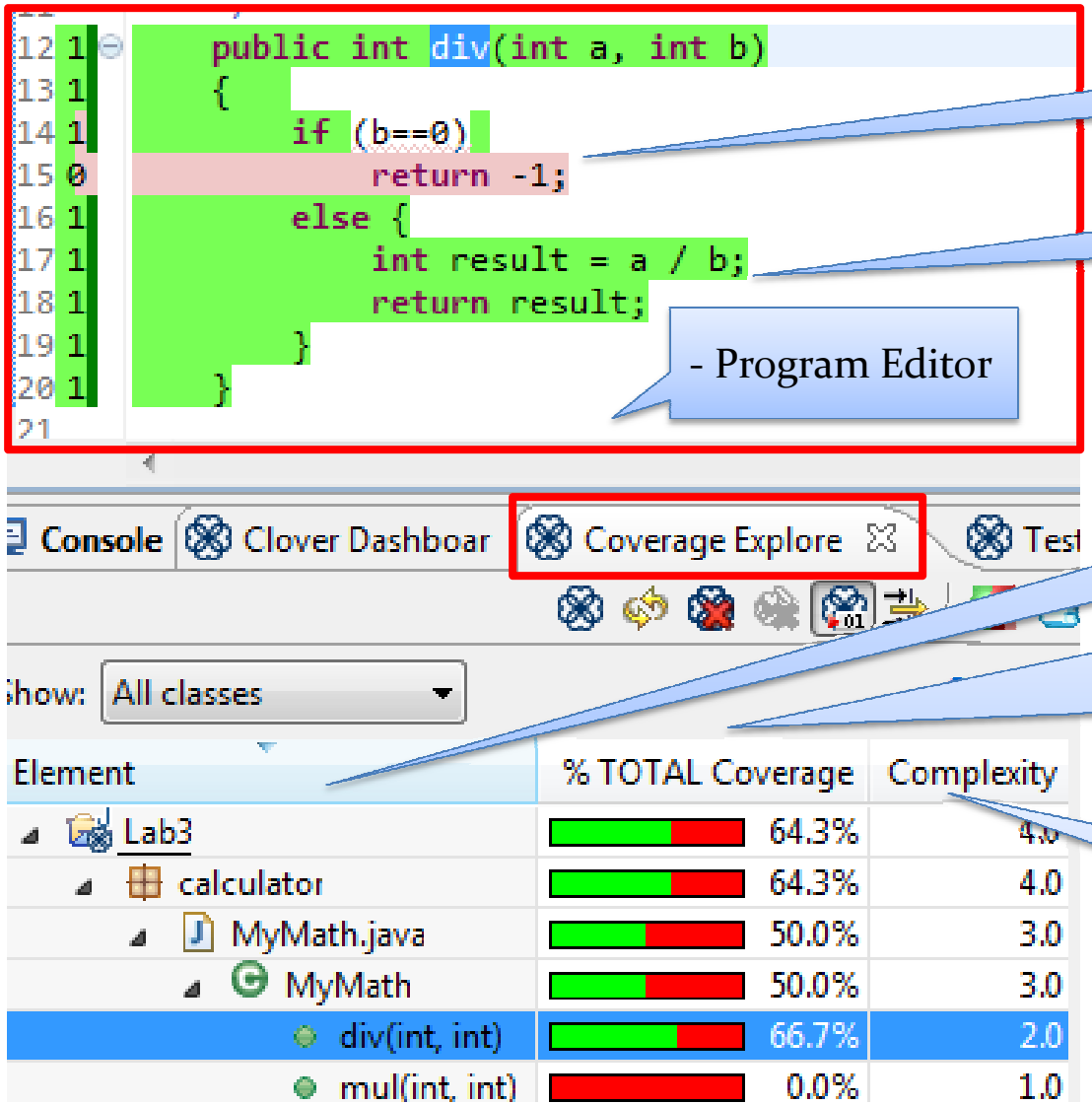


# (4). Clover Dashboard

- It's a place where you can have a quick overview of your project and find hints about code areas you should focus on:



# (1). Coverage Explorer




- Pink background used for uncovered code

- Green background used for covered code

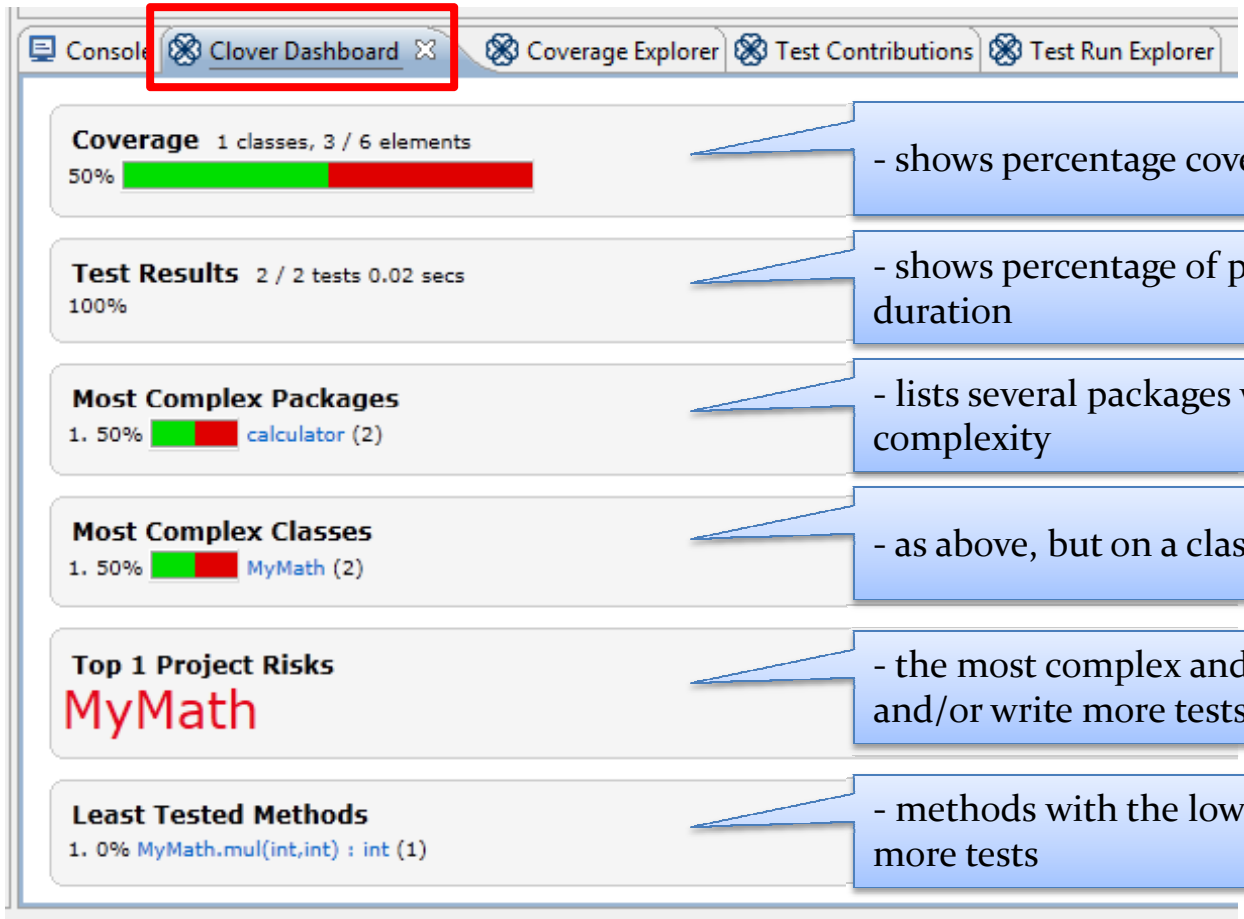
- Program Editor

- The name of the package, file, class or method.

-The total coverage of the element as a percentage.
- **Green bar** shows % of covered code
- **Red bar** shows % of uncovered code

-The complexity of the element
- For example, Line of Code (*LOC*) is major factor for high or low complexity of class. Complexity also depends on *LOC*

# (2). Clover Dashboard




- shows percentage coverage of the project

- shows percentage of passed tests as well as duration

- lists several packages with the highest complexity

- as above, but on a class level

- the most complex and the least tested classes and/or write more tests

- methods with the lowest test coverage -> write more tests

# Practical Example (Lab4) - *Details*

- **Application Classes**
  - ***MyComplexMath*** class contains only one method pow(double, double), pow() calculates the power of given numbers, e.g. pow (2, 4) = 16.
  - ***MySimpleMath*** class contains only one method div(int, int), div() calculates the division of given numbers, e.g. div (10, 5) = 2.

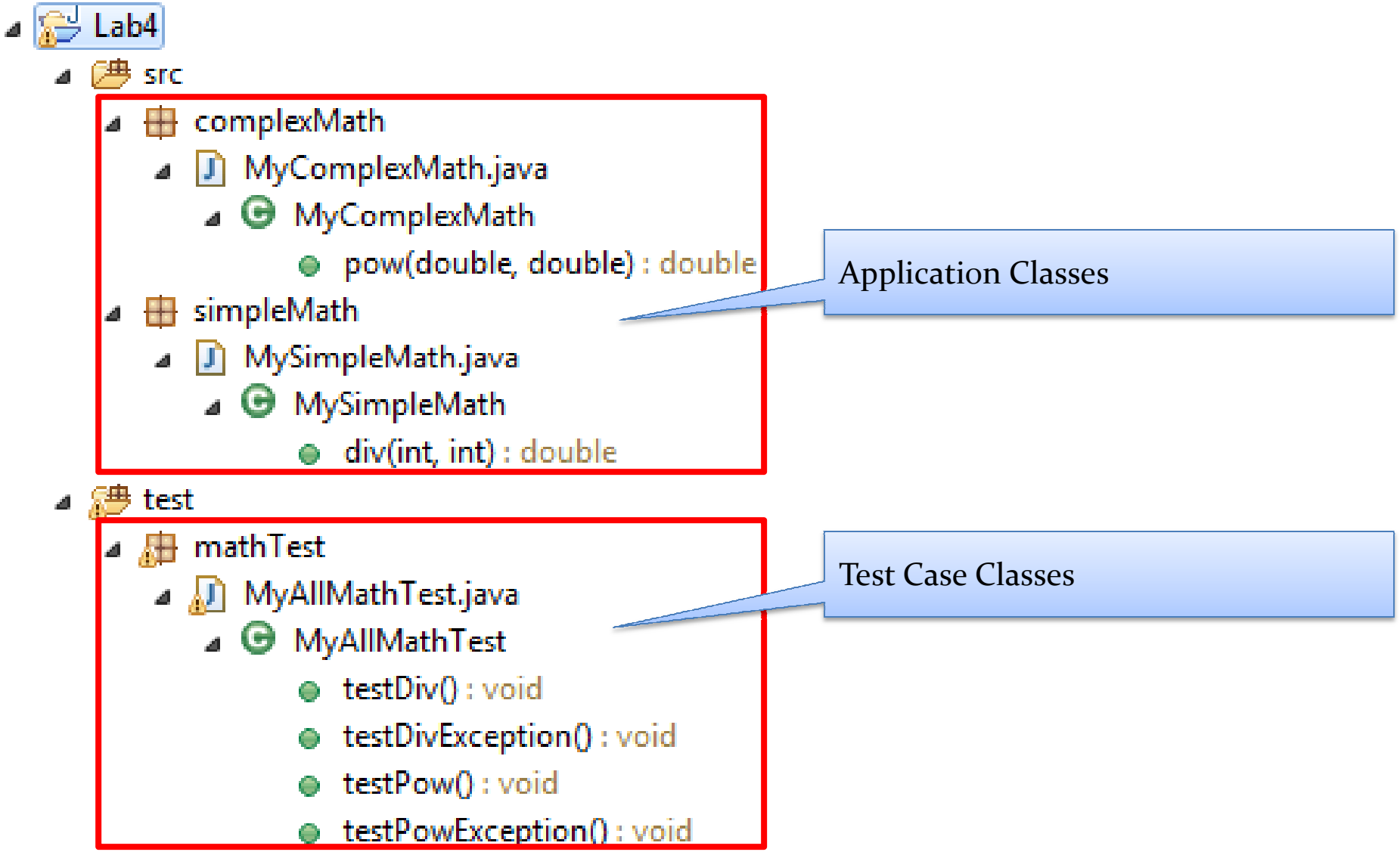
- **Implement the following Test Cases, i.e. TC1, TC2, TC3, TC4**

| MyComplexMath.pow (double, double) | | | |
|---|---|---|---|
| **TEST CASE** | **INPUT** | | **EXPECTED OUTPUT** |
| | **NUMBER1** | **NUMBER2** | **Power (NUMBER1, NUMBER2) = $X^n$ = ?** |
| TC-1 | 2.0 | 4.0 | 16.0 |
| TC-2 | 2.0 | 1024 | IllegalArgumentException |

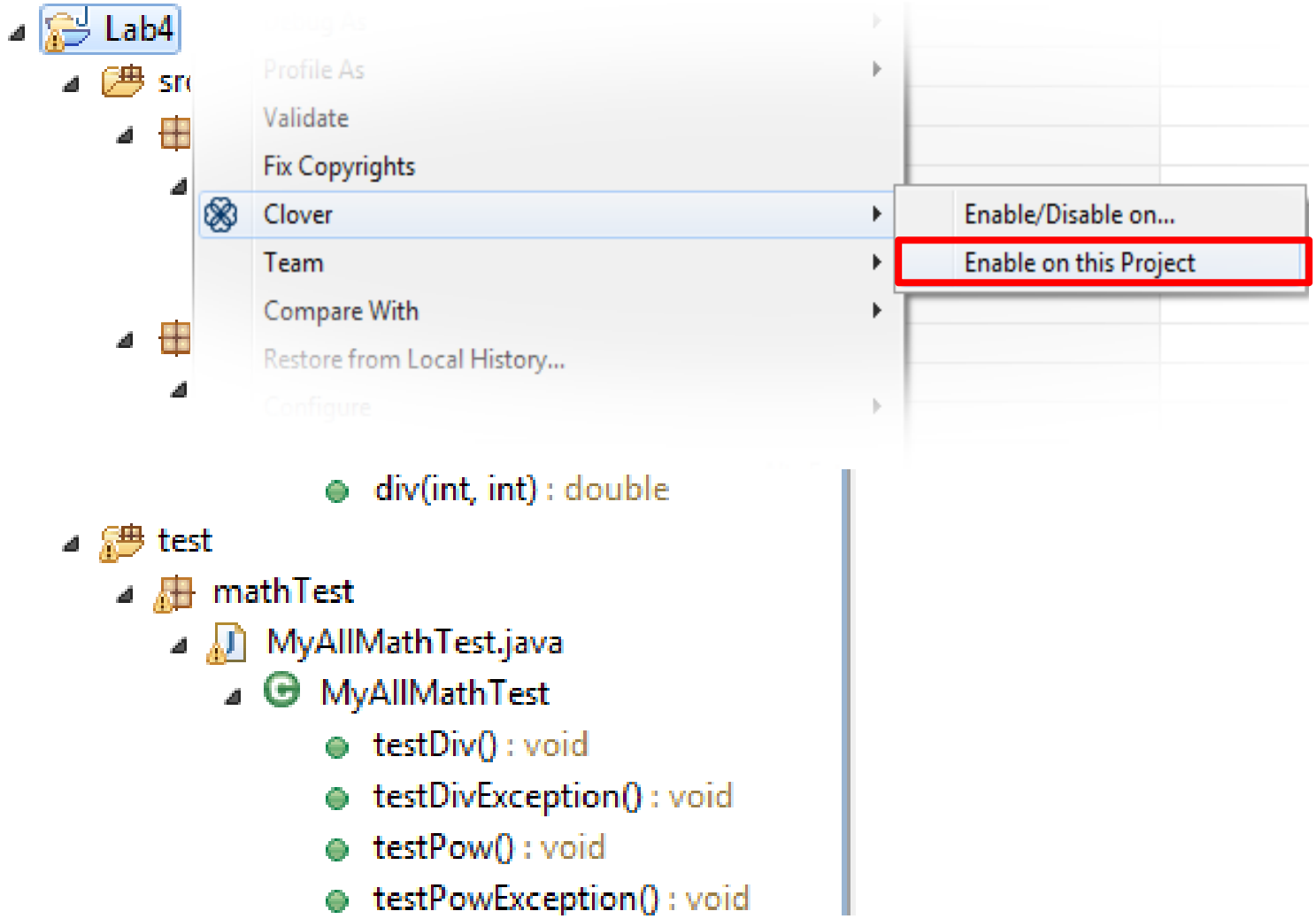
| MySimpleMath.div(int, int) | | | |
|---|---|---|---|
| **TEST CASE** | **INPUT** | | **EXPECTED OUTPUT** |
| | **NUMBER1** | **NUMBER2** | **NUMBER1 / NUMBER2 = ?** |
| TC-3 | 10 | 5 | 2.0 |
| TC-4 | 10 | 0 | IllegalArgumentException |

- **Zip project is available in lecture package**
  - Follow the steps by instructor to import the project in eclipse.

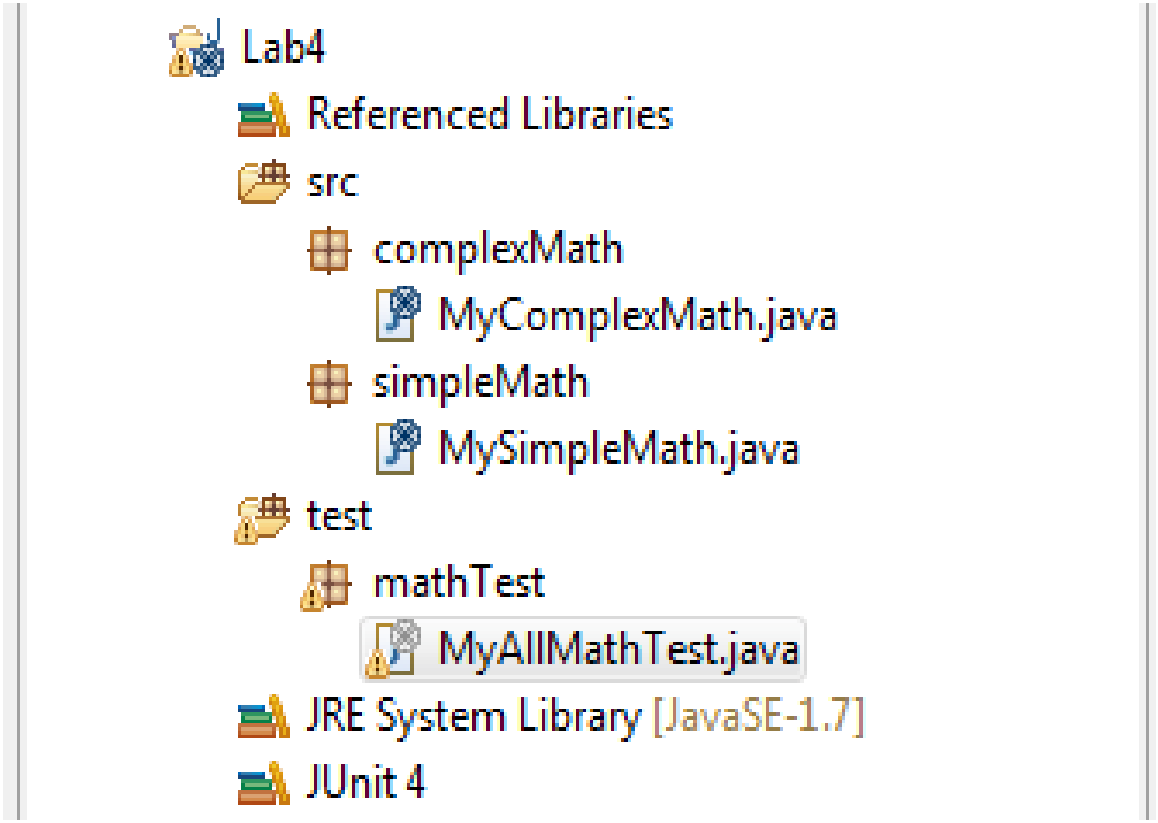
# Sample Project - *Details*



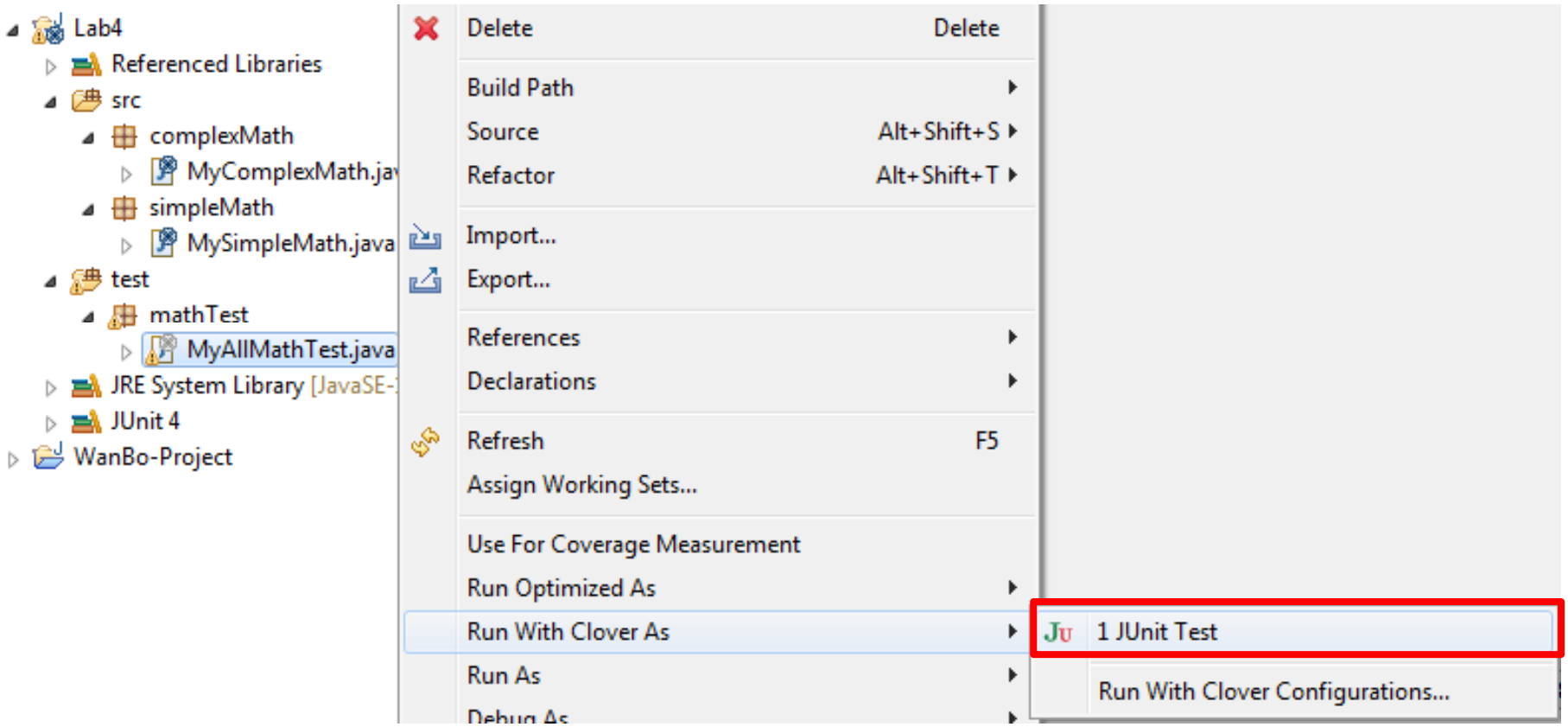
# Sample Project – *Clover Startup*



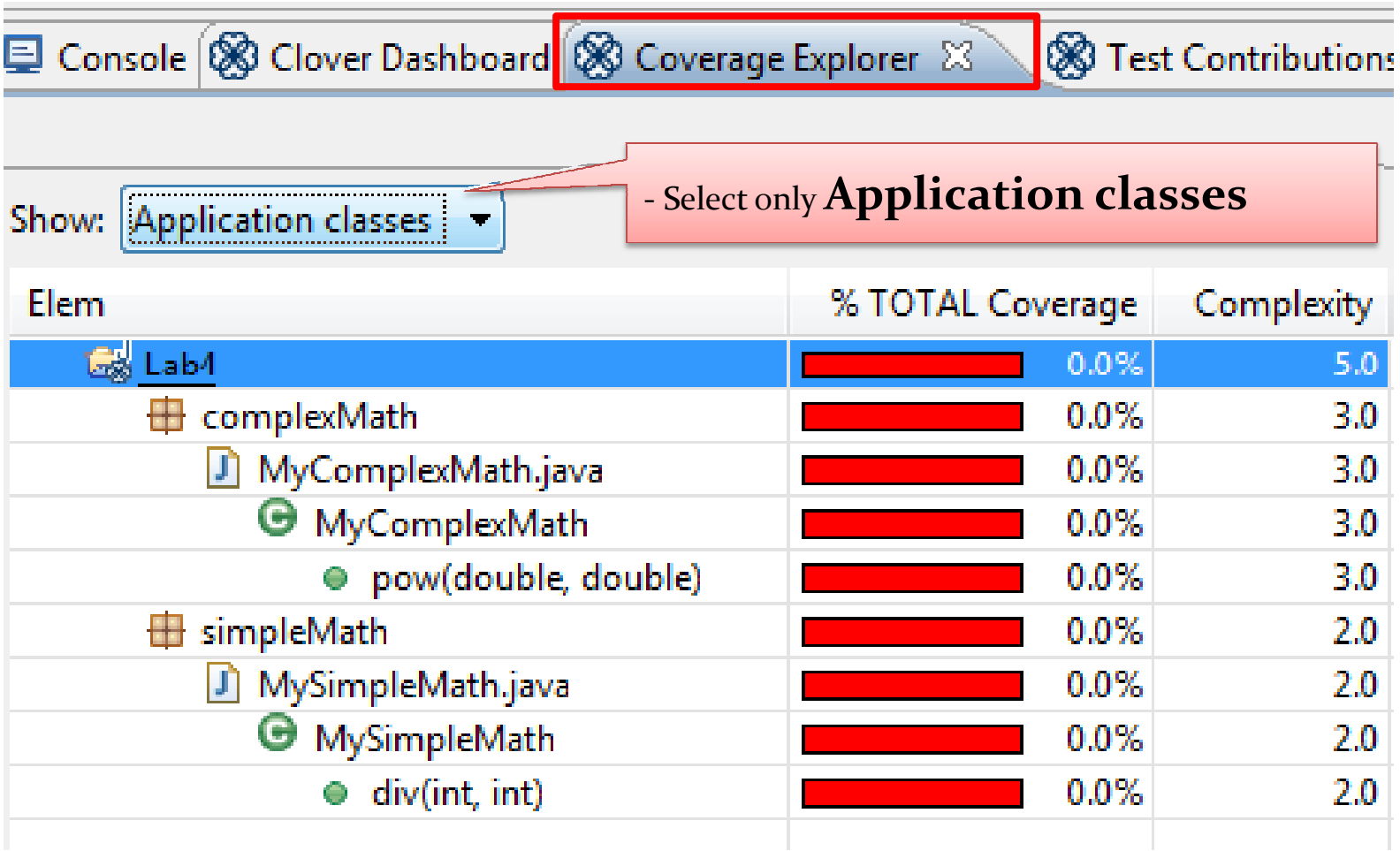
# Sample Project - *Clover Enabled*



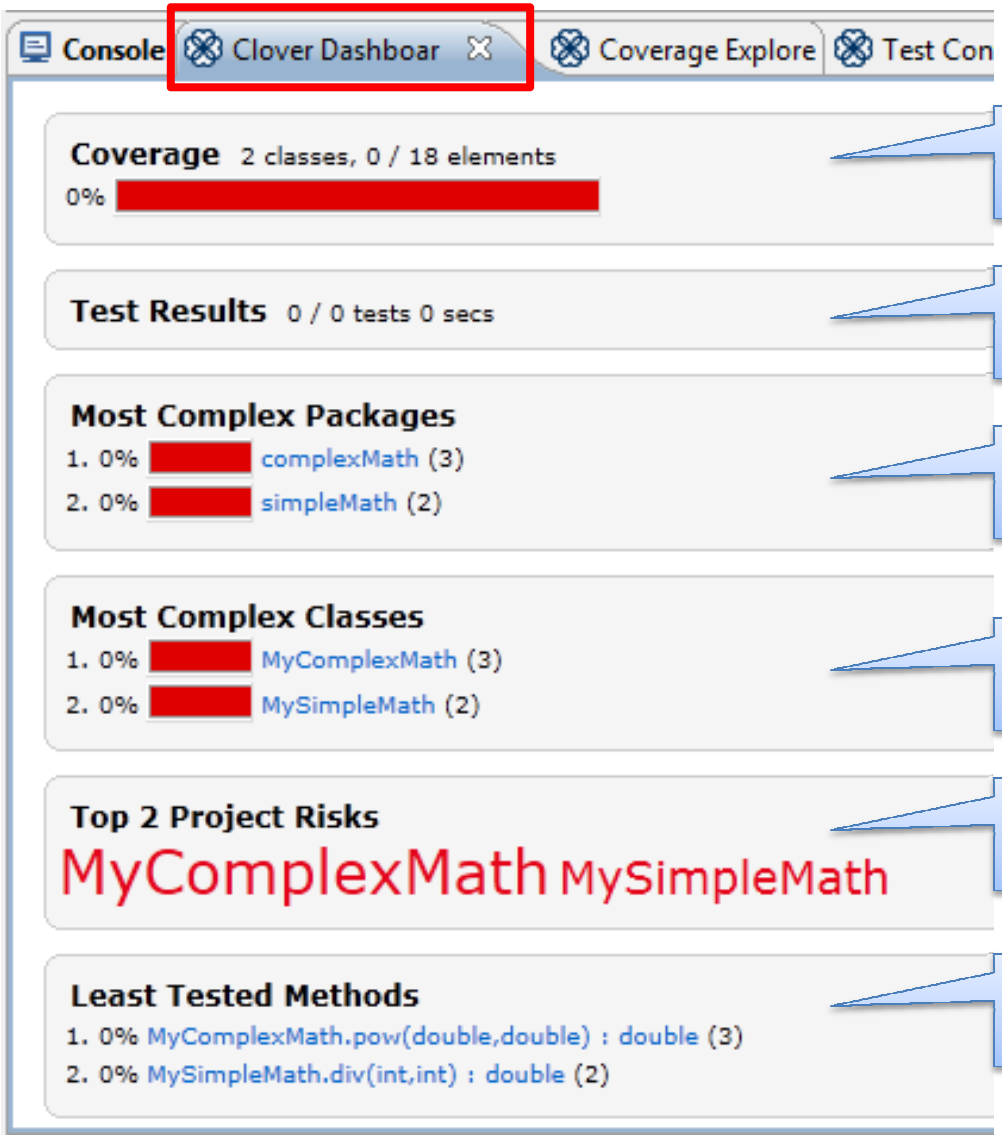
# Run With Clover As > *JUnit Test*



# Coverage is Zero, Why?



# Coverage is Zero, Why?




- shows percentage coverage of the project

- shows percentage of passed tests as well as duration

- lists several packages with the highest complexity

- as above, but on a class level

- the most complex and the least tested classes and/or write more tests

- methods with the lowest test coverage -> write more tests

# Test cases wasn't implemented

```
 9  public class MyAllMathTest {
10
11      @Test
12      public void testPow() {
13  //      Double a=2.0, n=4.0;
14  //      Double expected=Math.pow(a, n);
15  //      Double actual=MyComplexMath.pow(a, n);
16  //      assertEquals(expected, actual, 1);
17      }
18
19      @Test(expected=IllegalArgumentException.class)
20      public void testPowException() {
21  //      MyComplexMath.pow(Integer.MAX_VALUE, Integer.MAX_VALUE);
22      }
23
24      @Test
25      public void testDiv() {
26  //      Double expected=2.0;
27  //      Double actual=MySimpleMath.div(10, 5);
28  //      assertEquals(expected, actual, 1);
29      }
30
31      @Test(expected=IllegalArgumentException.class)
32      public void testDivException() {
33  //      MySimpleMath.div(10, 0);
34      }
35  }
```

# Zero % Coverage of MyComplexMath

```
3   public class MyComplexMath {
4
5 0     public static double pow(double a, double n) {
6 0         Double result=1.0;
7 0         for (int i=0; i<n; i++) {
8 0                 result*=a;
9 0                 if (result.isInfinite())
10 0                    throw new IllegalArgumentException();
11 0        }
12 0        return result;
13 0    }
14  }
```

- Code with pink background is not covered in test cases

Console | Clover Dashboard | Coverage Explorer | Test Contribution

how: Application classes

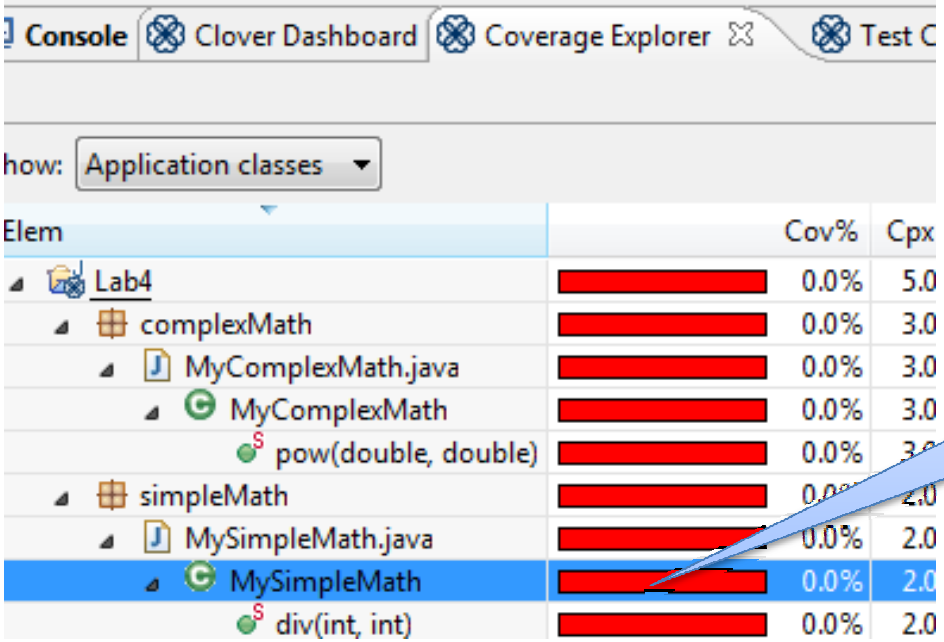
| Elem | Cov% | Cpx |
|---|---|---|
| Lab4 | 0.0% | 5.0 |
| complexMath | 0.0% | 3.0 |
| MyComplexMath.java | 0.0% | 3.0 |
| MyComplexMath | 0.0% | 3.0 |
| pow(double, double) | 0.0% | 3.0 |
| simpleMath | 0.0% | 2.0 |
| MySimpleMath.java | 0.0% | 2.0 |
| MySimpleMath | 0.0% | 2.0 |
| div(int, int) | 0.0% | 2.0 |

- Zero % Coverage

# Zero % Coverage of MySimpleMath

```
3   public class MySimpleMath {
4
5 0     public static double div (int a, int b) {
6 0         if (b==0)
7 0             throw new IllegalArgumentException();
8 0         else {
9 0             double result = a / b;
10 0            return result;
11 0        }
12 0
13 0    }
14  }
```

- Code with pink background is not covered in test cases




- Zero % Coverage

# Selecting the Test cases for Application Classes

## Solution

| MyComplexMath.pow (double, double) | | | |
|---|---|---|---|
| TEST CASE | INPUT | | EXPECTED OUTPUT |
| | NUMBER1 | NUMBER2 | Power (NUMBER1, NUMBER2) = ? |
| TC-1 | 2.0 | 4.0 | 16.0 |

| MySimpleMath.div(int, int) | | | |
|---|---|---|---|
| TEST CASE | INPUT | | EXPECTED OUTPUT |
| | NUMBER1 | NUMBER2 | NUMBER1 / NUMBER2 = ? |
| TC-3 | 10 | 5 | 2.0 |

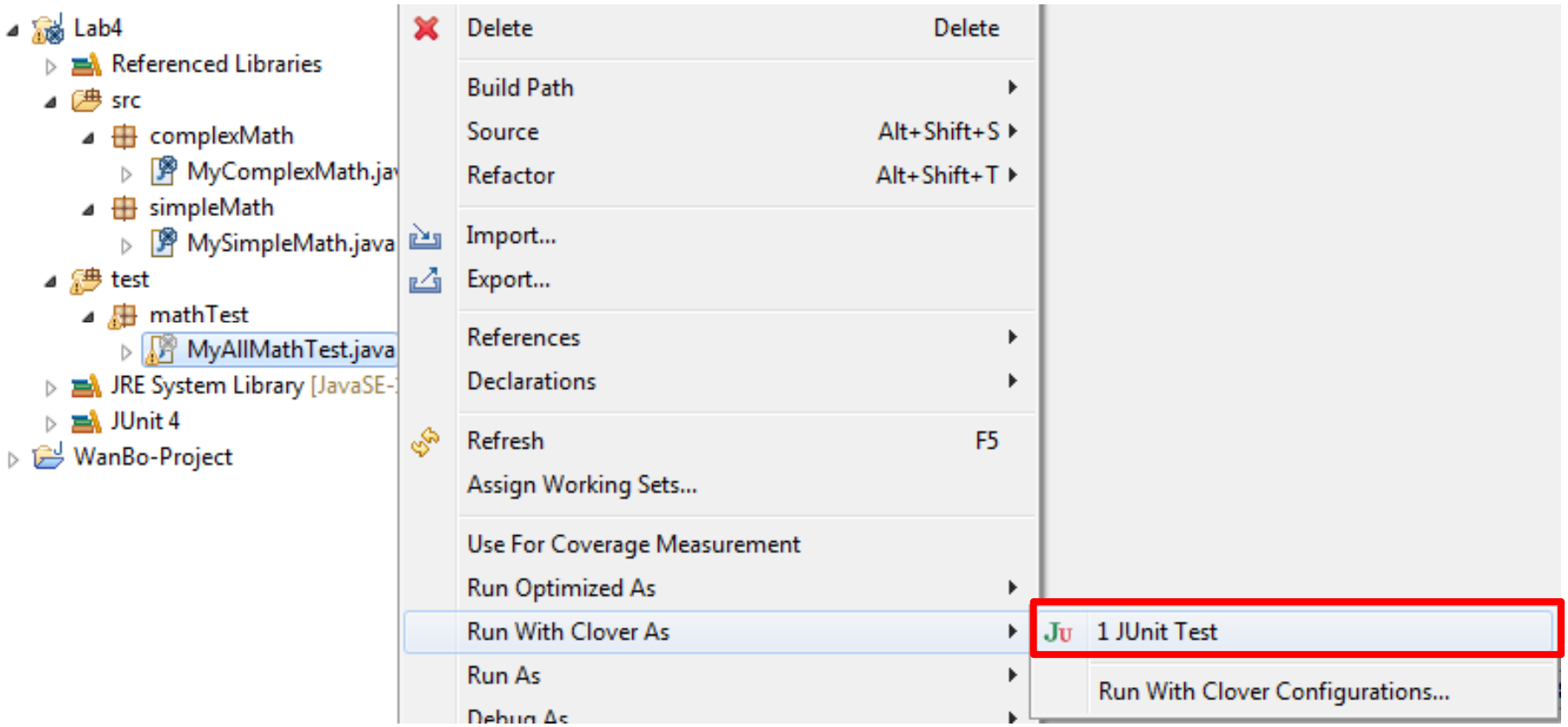
# Two (2) test cases implemented

```
 9  public class MyAllMathTest {
10
11      @Test
12      public void testPow() {
13          Double a=2.0, n=4.0;
14          Double expected=Math.pow(a, n);
15          Double actual=MyComplexMath.pow(a, n);
16          assertEquals(expected, actual, 1);
17      }
18
19      @Test(expected=IllegalArgumentException.class)
20      public void testPowException() {
21  //          MyComplexMath.pow(Integer.MAX_VALUE, Integer.MAX_VALUE);
22      }
23
24      @Test
25      public void testDiv() {
26          Double expected=2.0;
27          Double actual=MySimpleMath.div(10, 5);
28          assertEquals(expected, actual, 1);
29      }
30
31      @Test(expected=IllegalArgumentException.class)
32      public void testDivException() {
33  //          MySimpleMath.div(10, 0);
34      }
35  }
```

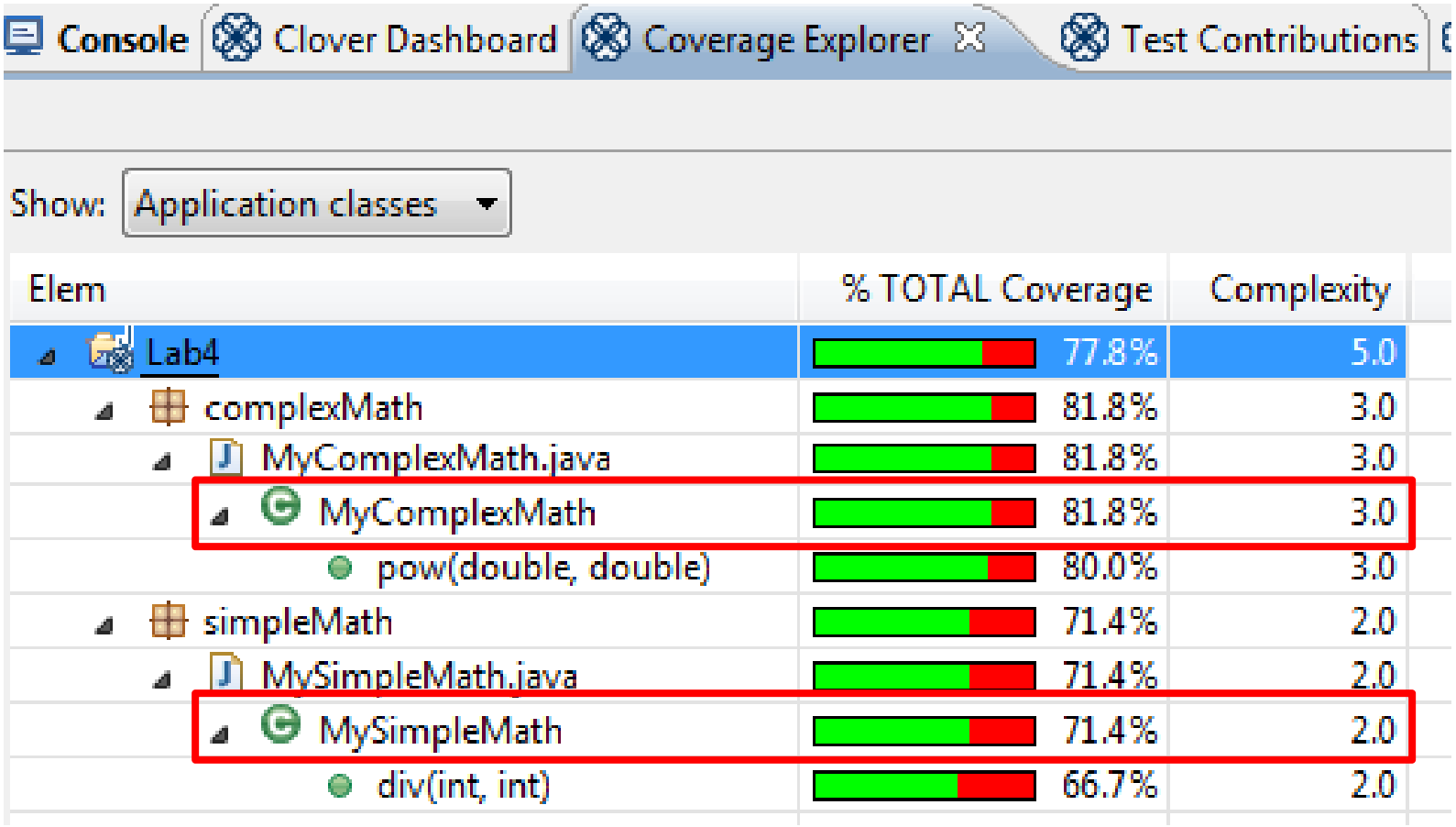
- Remove the comments on line # 13-16

- Remove the comments on line # 26-28

# Re-run With Clover As > *JUnit Test*



# Clover Code Coverage Explorer *Results*



# Reviewing the MyComplexMath

```
3   public class MyComplexMath {
4
5 1     public double pow(double a, double n) {
6 1         Double result=1.0;
7 1         for (int i=0; i<n; i++) {
8 4             result*=a;
9 4             if (result.isInfinite())
10 0                throw new IllegalArgumentException();
11 1        }
12 1        return result;
13 1    }
14  }
```

| MyComplexMath.pow (double, double) | | | |
|---|---|---|---|
| **TEST CASE** | **INPUT** | | **EXPECTED OUTPUT** |
| | **NUMBER1** | **NUMBER2** | **Power (NUMBER1, NUMBER2) = ?** |
| TC-2 | 2.0 | 1024 | IllegalArgumentException |

# Test case implementation

```
 9  public class MyAllMathTest {
10
11      @Test
12      public void testPow() {
13          Double a=2.0, n=4.0;
14          Double expected=Math.pow(a, n);
15          Double actual=MyComplexMath.pow(a, n);
16          assertEquals(expected, actual, 1);
17      }
18
19      @Test(expected=IllegalArgumentException.class)
20      public void testPowException() {
21          MyComplexMath.pow(2.0, 1024);
22      }
```

- Remove the comments on line # 21

# Re-run the *Test*

```
3       public class MyComplexMath {
4
5     7     public static double pow(double a, double n) {
6     7         Double result=1.0;
7     7         for (int i=0; i<n; i++) {
8  1066                 result*=a;
9  1066                 if (result.isInfinite())
10    2                     throw new IllegalArgumentException();
11    7         }
12    5         return result;
13    7     }
14      }
```

Console | Clover Dashboard | Coverage Explorer | Test Contributions

Show: Application classes

| Elem | % TOTAL Coverage | Complexity |
|---|---|---|
| Lab4 | 88.9% | 5.0 |
| complexMath | 100.0% | 3.0 |
| MyComplexMath.java | 100.0% | 3.0 |
| MyComplexMath | 100.0% | 3.0 |
| pow(double, double) | 100.0% | 3.0 |
| simpleMath | 71.4% | 2.0 |
| MySimpleMath.java | 71.4% | 2.0 |
| MySimpleMath | 71.4% | 2.0 |
| div(int, int) | 66.7% | 2.0 |

**100 % Coverage**

# Reviewing the MySimpleMath

```
3     public class MySimpleMath {
4
5  1      public double div (int a, int b) {
6  1          if (b==0)
7  0              throw new IllegalArgumentException();
8  1          else {
9  1              int result = a / b;
10 1              return result;
11 1          }
12 1
13 1      }
14    }
```

| MySimpleMath.div (int, int) | | | |
|---|---|---|---|
| **TEST CASE** | **INPUT** | | **EXPECTED OUTPUT** |
| | **NUMBER1** | **NUMBER2** | **Power (NUMBER1, NUMBER2) = ?** |
| TC-4 | 10 | 0 | IllegalArgumentException |

# Test case implementation

```
 9  public class MyAllMathTest {
10
11      @Test
12      public void testPow() {
13          Double a=2.0, n=4.0;
14          Double expected=Math.pow(a, n);
15          Double actual=MyComplexMath.pow(a, n);
16          assertEquals(expected, actual, 1);
17      }
18
19      @Test(expected=IllegalArgumentException.class)
20      public void testPowException() {
21          MyComplexMath.pow(2.0, 1024);
22      }
23
24      @Test
25      public void testDiv() {
26          Double expected=2.0;
27          Double actual=MySimpleMath.div(10, 5);
28          assertEquals(expected, actual, 1);
29      }
30
31      @Test(expected=IllegalArgumentException.class)
32      public void testDivException() {
33          MySimpleMath.div(10, 0);
34      }
35  }
```

- Remove the comments on line # 33

# Re-run the *Test*

```
3   public class MySimpleMath {
4
5 6     public static double div (int a, int b) {
6 6         if (b==0)
7 1             throw new IllegalArgumentException();
8 6         else {
9 5             double result = a / b;
10 5            return result;
11 6        }
12 6
13 6    }
14  }
```

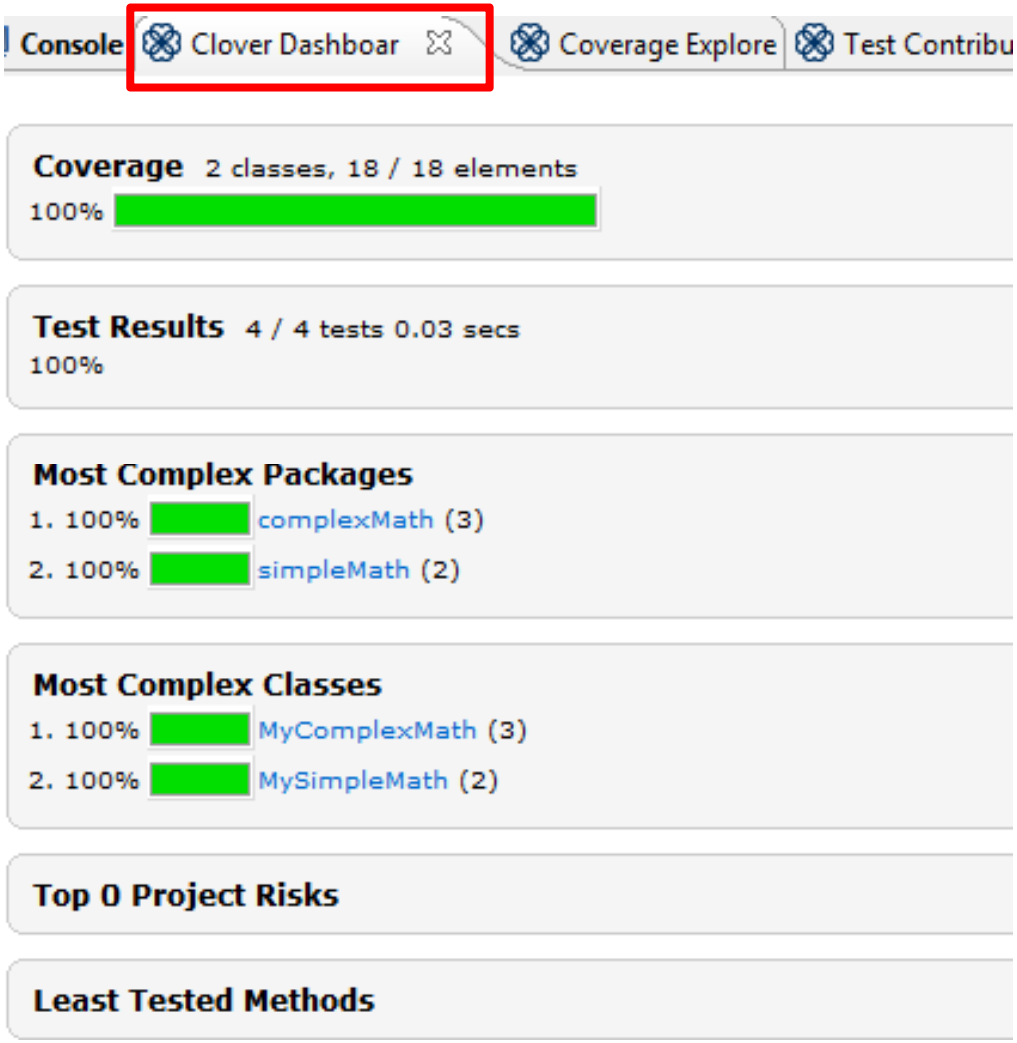
Console | Clover Dashboard | Coverage Explorer | Test Contributior

Show: Application classes

| Elem | % TOTAL Coverage | Complexity |
|---|---|---|
| Lab4 | 100.0% | 5.0 |
| complexMath | 100.0% | 3.0 |
| MyComplexMath.java | 100.0% | 3.0 |
| MyComplexMath | 100.0% | 3.0 |
| pow(double, double) | 100.0% | 3.0 |
| simpleMath | 100.0% | 2.0 |
| MySimpleMath.java | 100.0% | 2.0 |
| MySimpleMath | 100.0% | 2.0 |
| div(int, int) | 100.0% | 2.0 |

100 % Coverage

# Clover Dashboard




100 % Coverage

# SWE 434
# Software Testing and Validation

## Code Coverage Analysis by Clover

*Lecture slides are available on following website through your university student login*
https://lms.ksu.edu.sa/

**Raja Majid Mehmood**
rmehmood@ksu.edu.sa
Department of Software Engineering,
King Saud University, Riyadh, Saudi Arabia.

**Some of the material in these slides is derived from https://confluence.atlassian.com/, Thanks to them and also special thanks to Dr. Shamim**

# Lecture's Agenda

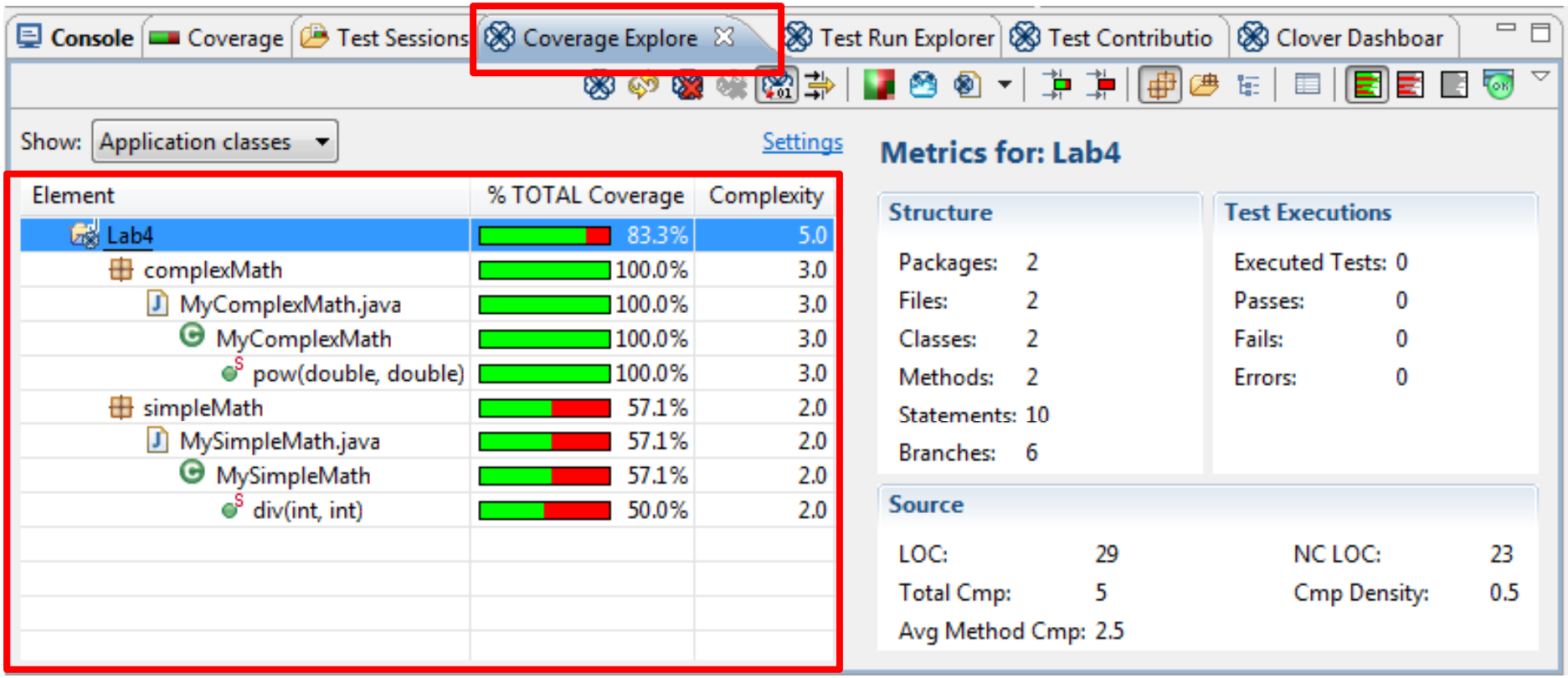
1. Briefly discuss the Clover Explorer

2. How to analyze the **Code Coverage Elements**
   1. *Branch*
   2. *Statement*
   3. *Method*

3. **Quiz – 2**

4. For more details, check the *Clover* website
   – https://confluence.atlassian.com/display/CLOVER/2.+Exploration+of+coverage+in+Eclipse

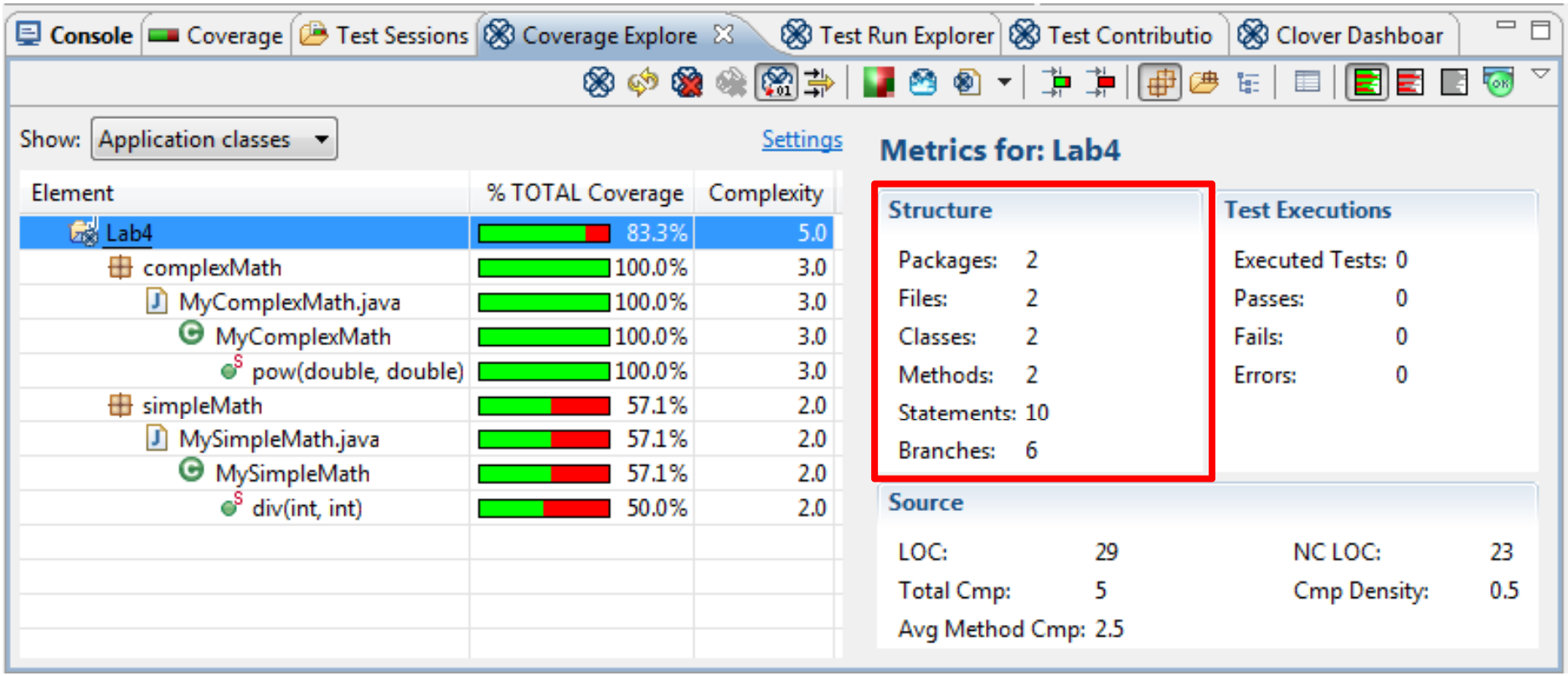
# Coverage Explorer




The Coverage Explorer's tree displays the following columns:

- **Element** - The name of the package, file, class or method.
- **%TOTAL Coverage** - The total coverage of the element as a percentage.
- **Complexity** – The complexity of the element

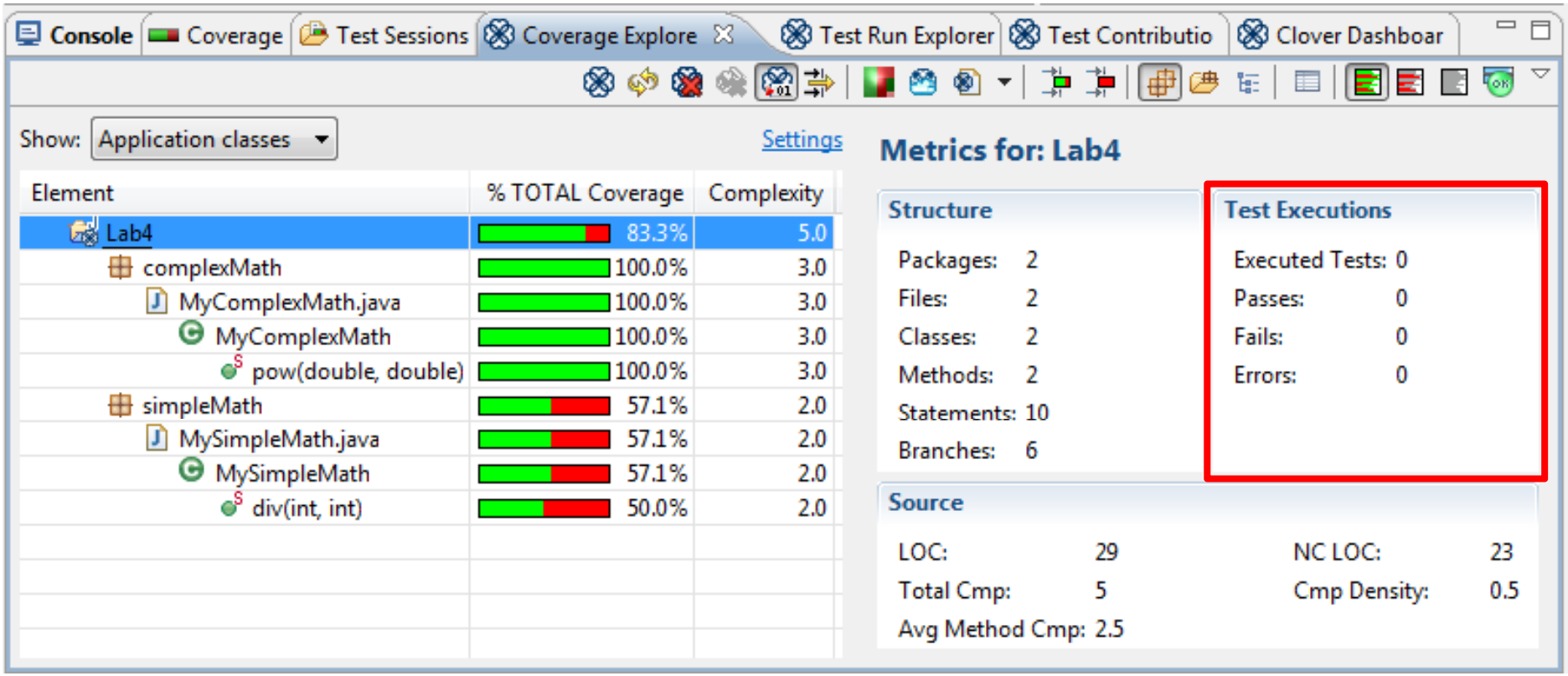
# Coverage Explorer




## Structure

**Packages:** The number of packages the project or package root contains
**Files:** The number of files the package, package root or project contains
**Classes:** The number of classes the file, package, package root or project contains
**Methods:** The number of methods the class, file, package, package root or project contains
**Statements:** The number of statements the method, class, file, package, package root or project contains
**Branches:** The number of branches the method, class, file, package, package root or project contains

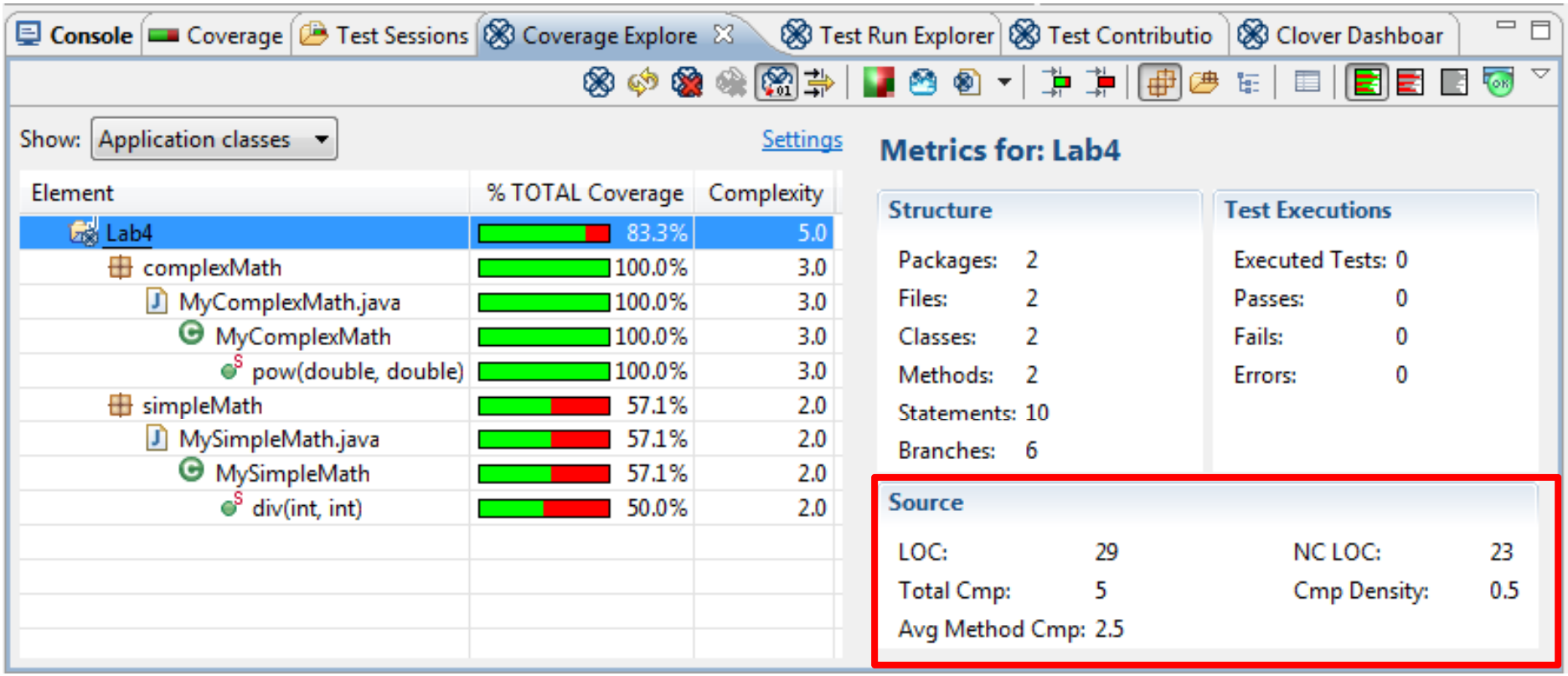
# Coverage Explorer




## Tests:

**Tests:** The total number of tests in the class, file, package, package root or project
**Passes:** The total number of passing tests in the class, file, package, package root or project
**Fails:** The total number of failing tests in the class, file, package, package root or project
**Errors:** The total number of tests with errors in the class, file, package, package root or project. Currently Clover does not differentiate between fails and errors and counts all errors as failures.

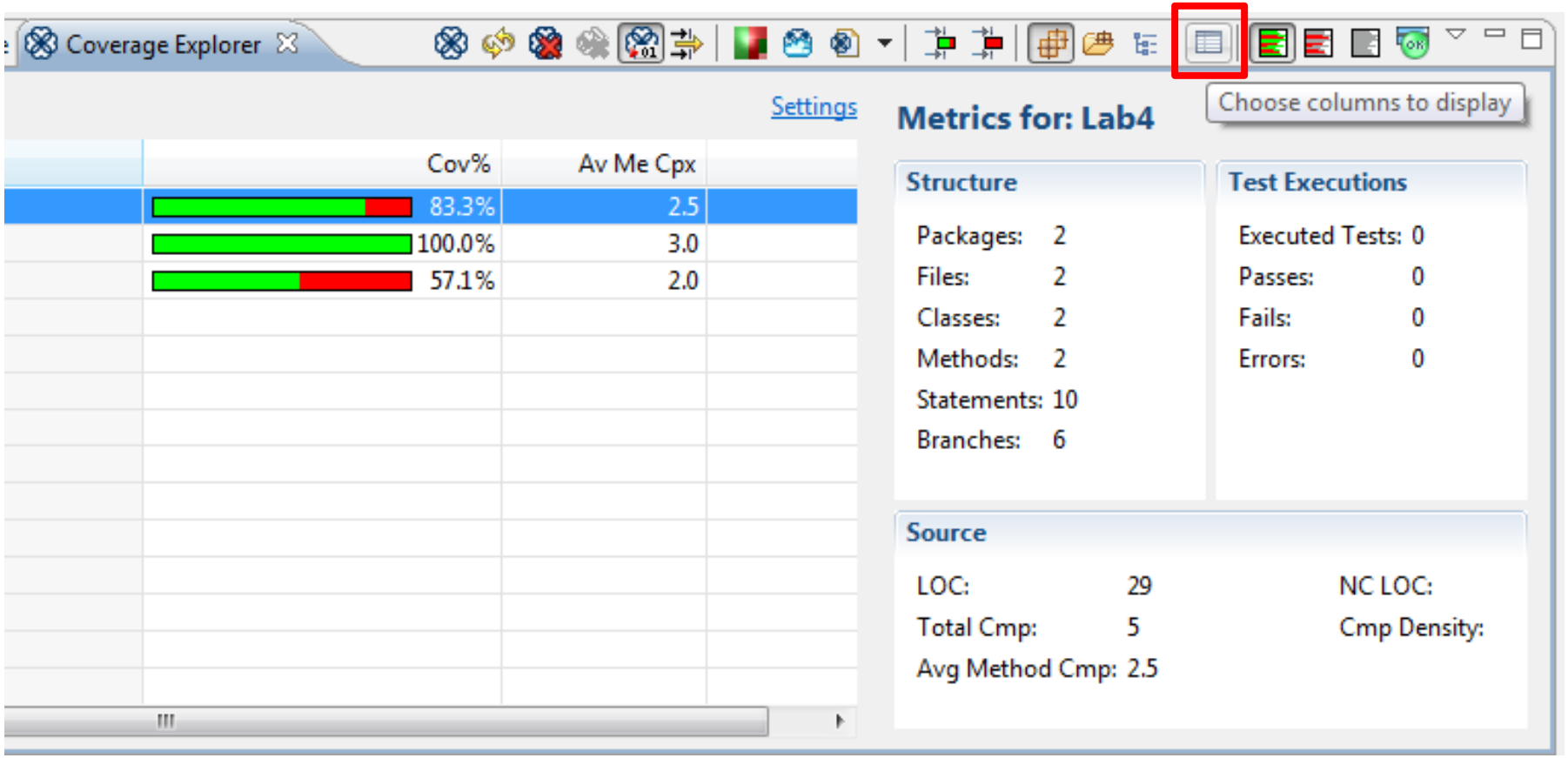
# Coverage Explorer



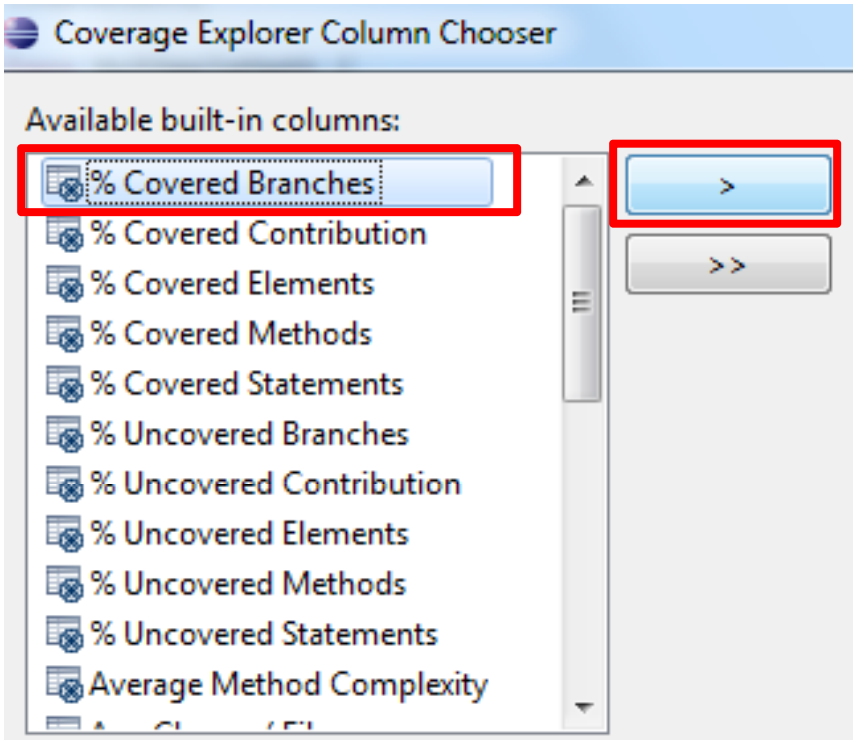

## Source:

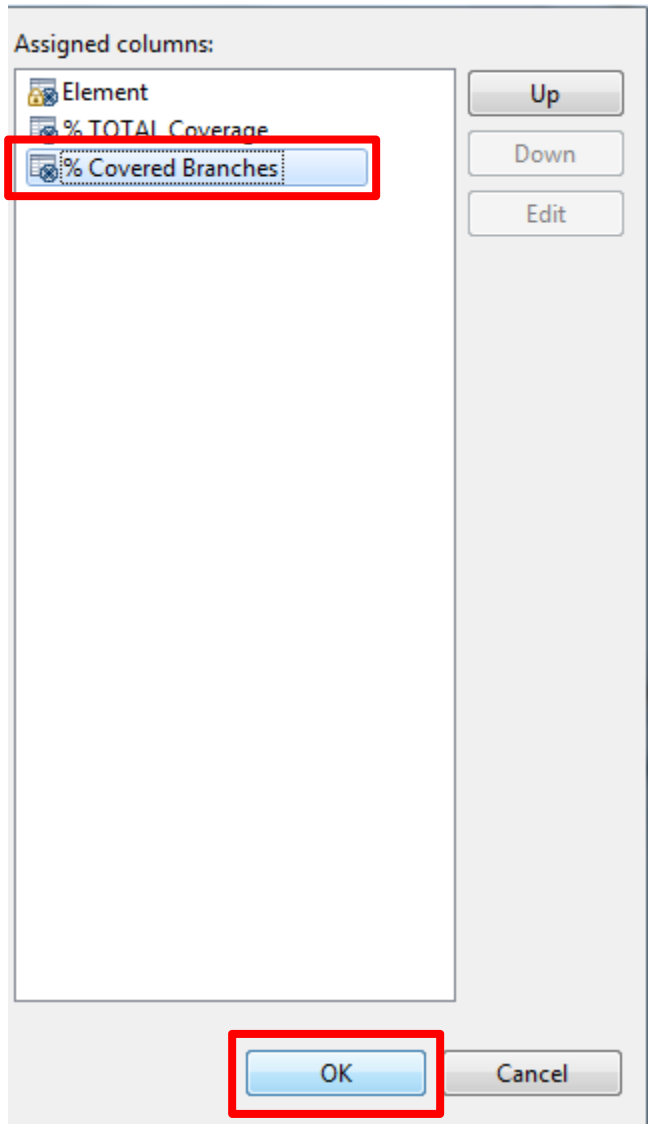
**LOC:** The total number of lines of code in the class, file, package, package root or project
**NC LOC:** The total number of non-commented lines of code in the class, file, package, package root or project
**Total Cmp:** The total complexity of code in the method, class, file, package, package root or project
**Avg Method Cmp:** The average method complexity of the class, file, package, package root or project
**Cmp Density:** The complexity density of code in the class, file, package, package root or project

# Columns

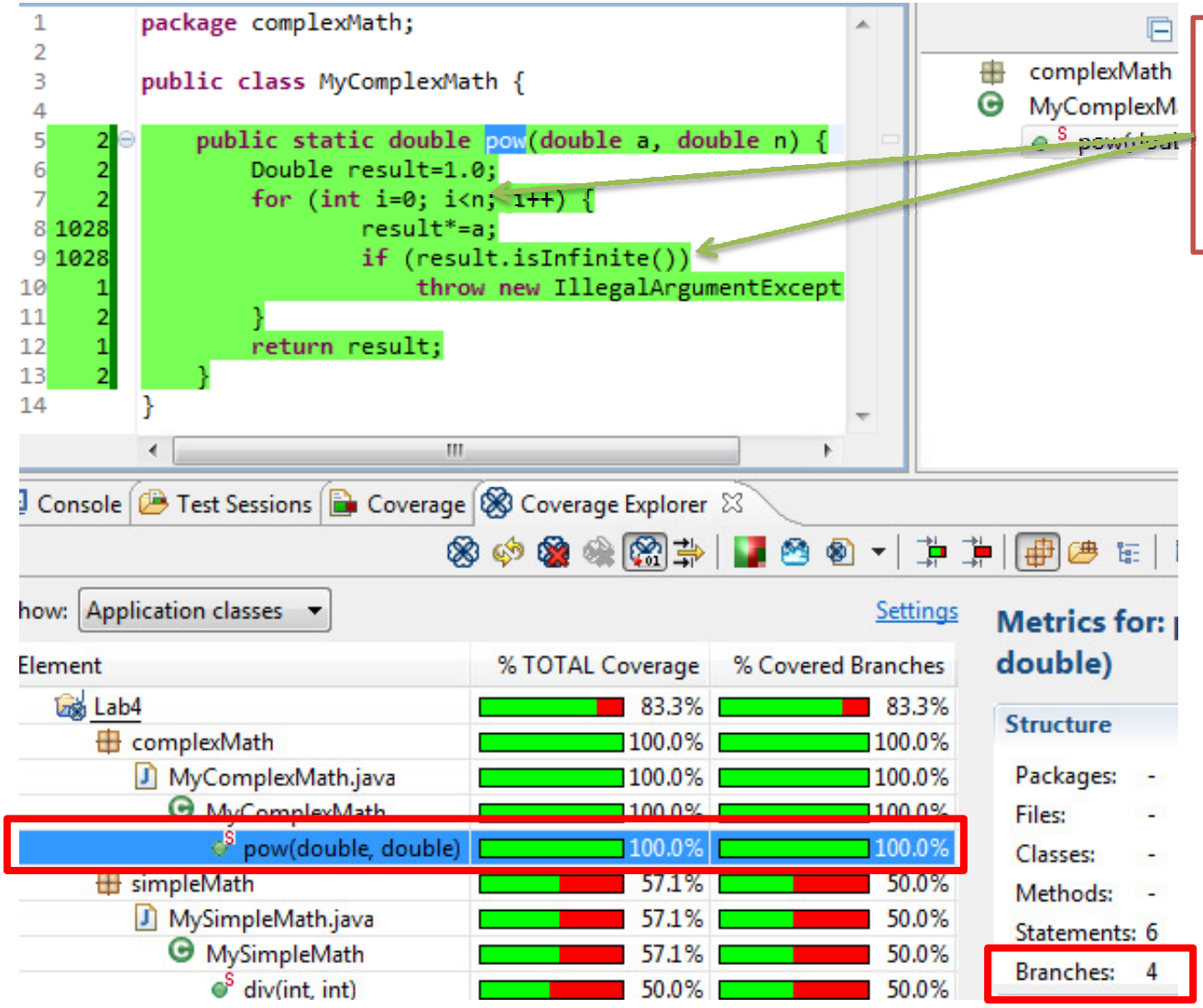

Coverage Explorer
Settings
Choose columns to display
Metrics for: Lab4
Cov%
Av Me Cpx
83.3%
2.5
100.0%
3.0
57.1%
2.0
Structure
Packages: 2
Files: 2
Classes: 2
Methods: 2
Statements: 10
Branches: 6
Test Executions
Executed Tests: 0
Passes: 0
Fails: 0
Errors: 0
Source
LOC: 29
NC LOC:
Total Cmp: 5
Cmp Density:
Avg Method Cmp: 2.5

# Selecting %Covered Branches



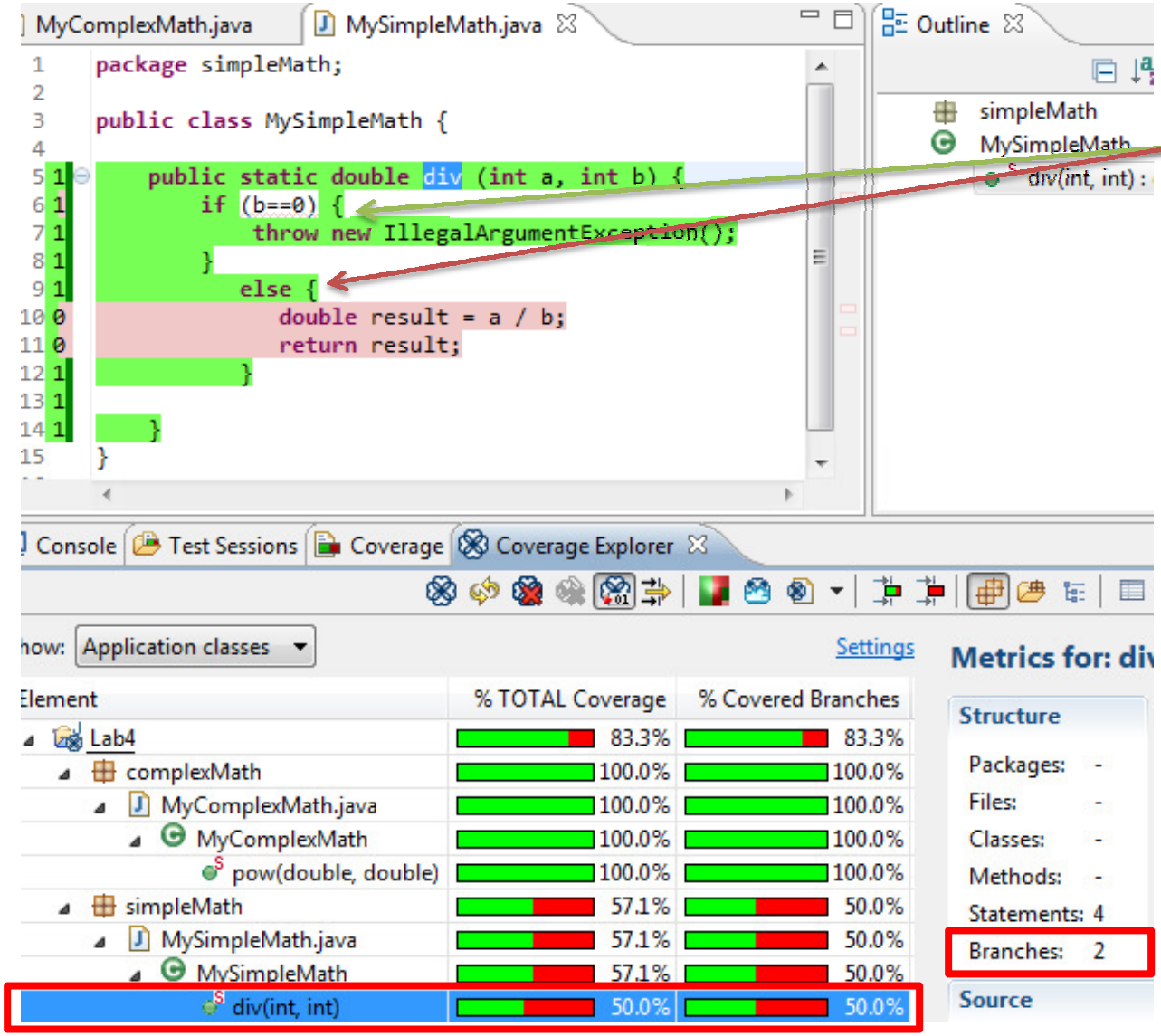



# 100% Branch Coverage



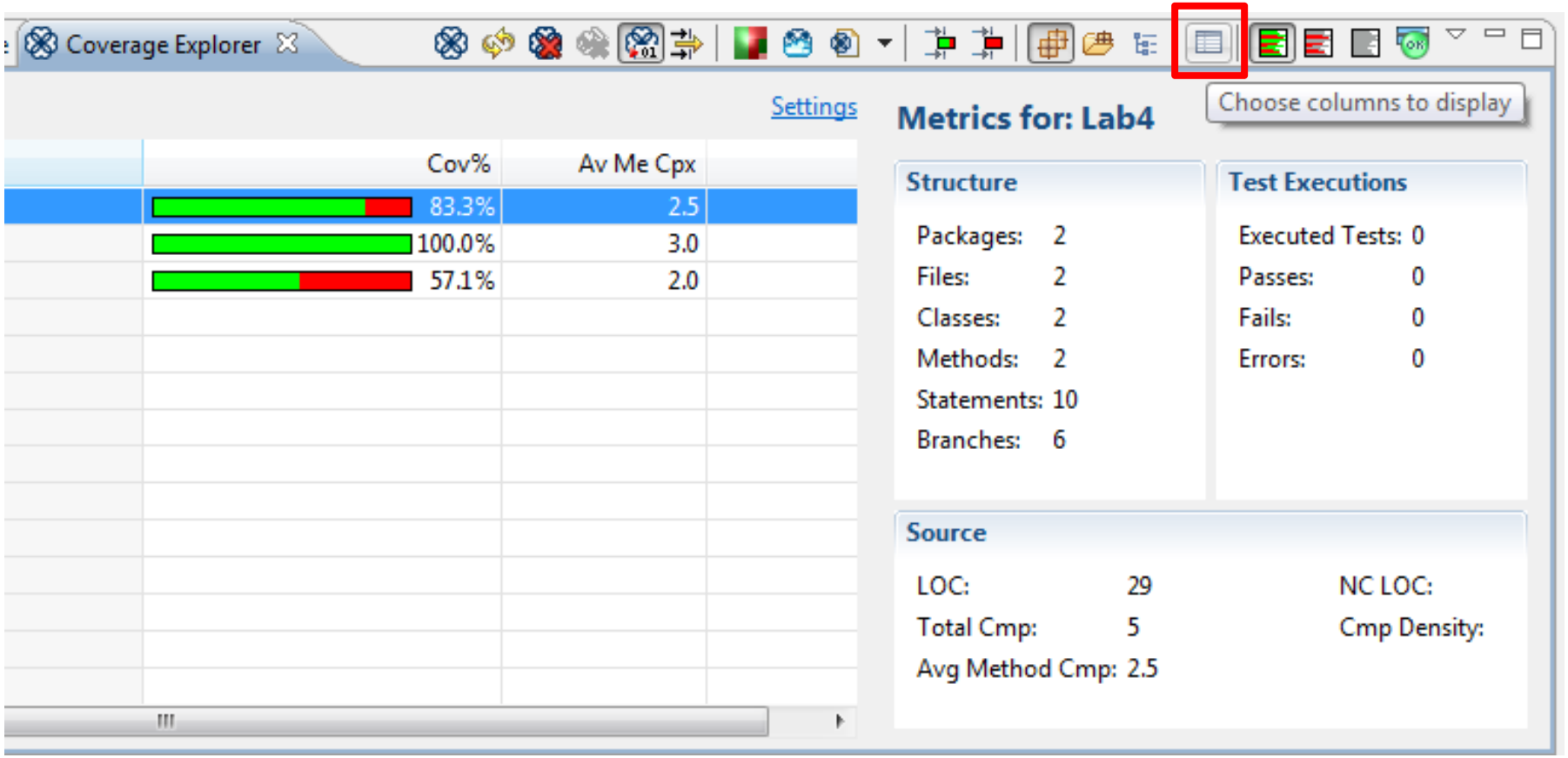

All branches covered
**Total Branches:** 4
**Covered Braches:** 4
**Uncovered Branches:** 0

# 50% Branch Coverage

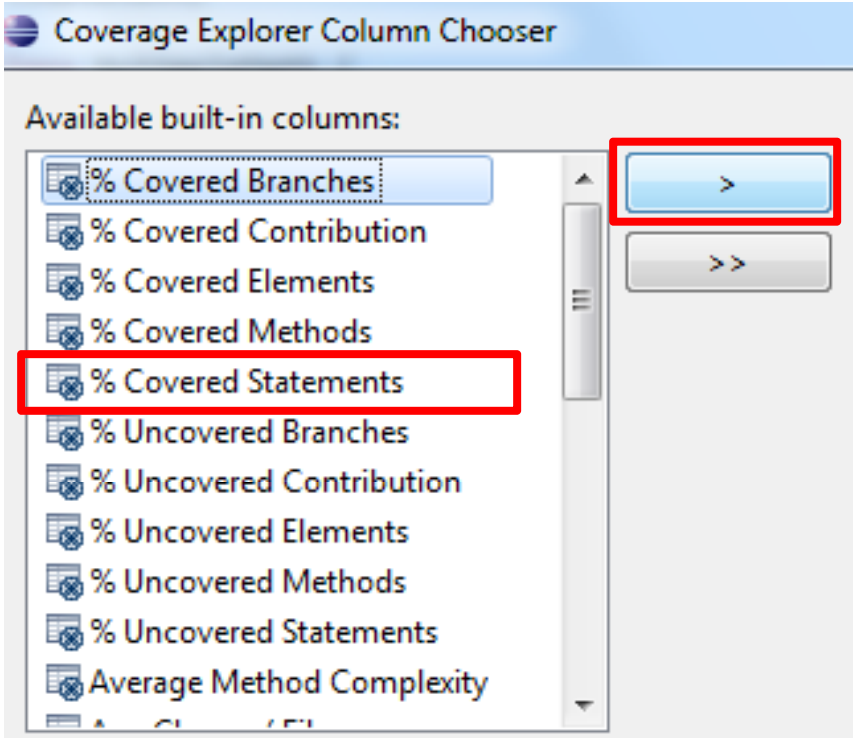


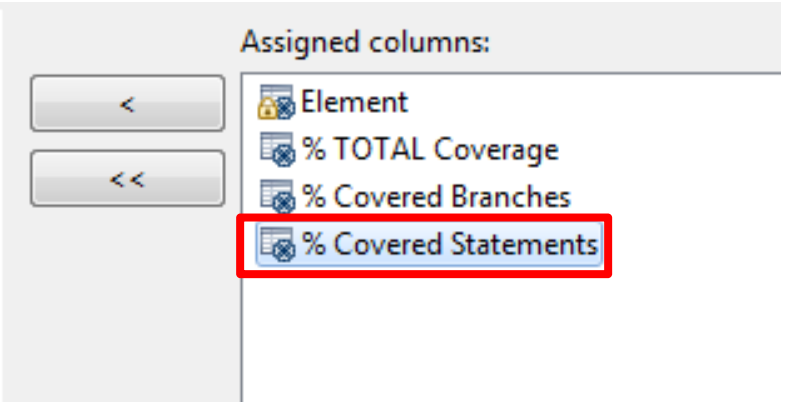

**Total Branches:** 2
**Covered Braches:** 1
**Uncovered Branches:** 1

# **More** *Columns (Adding more coverage elements)*

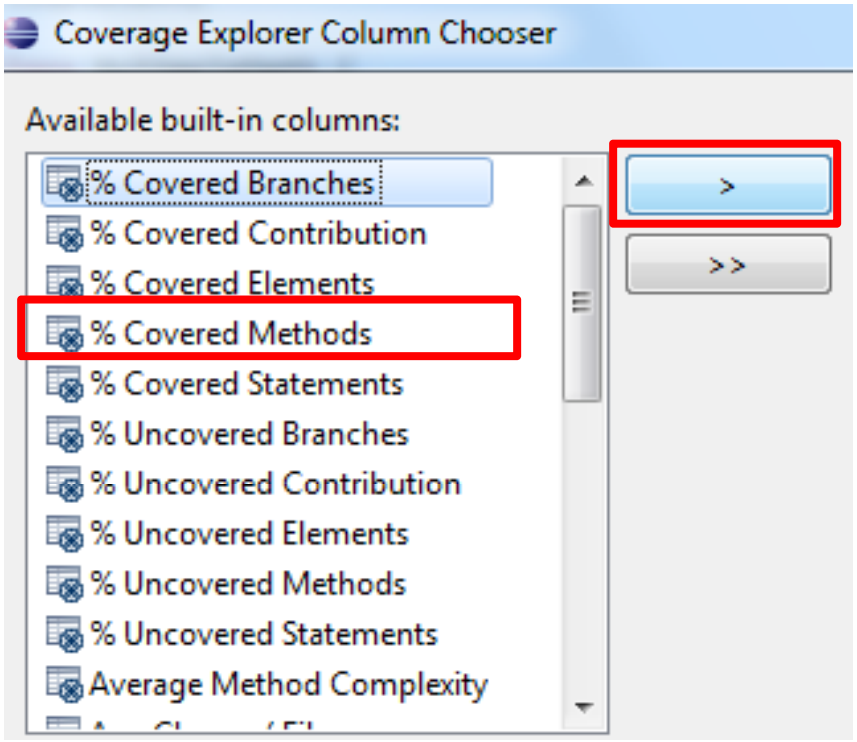


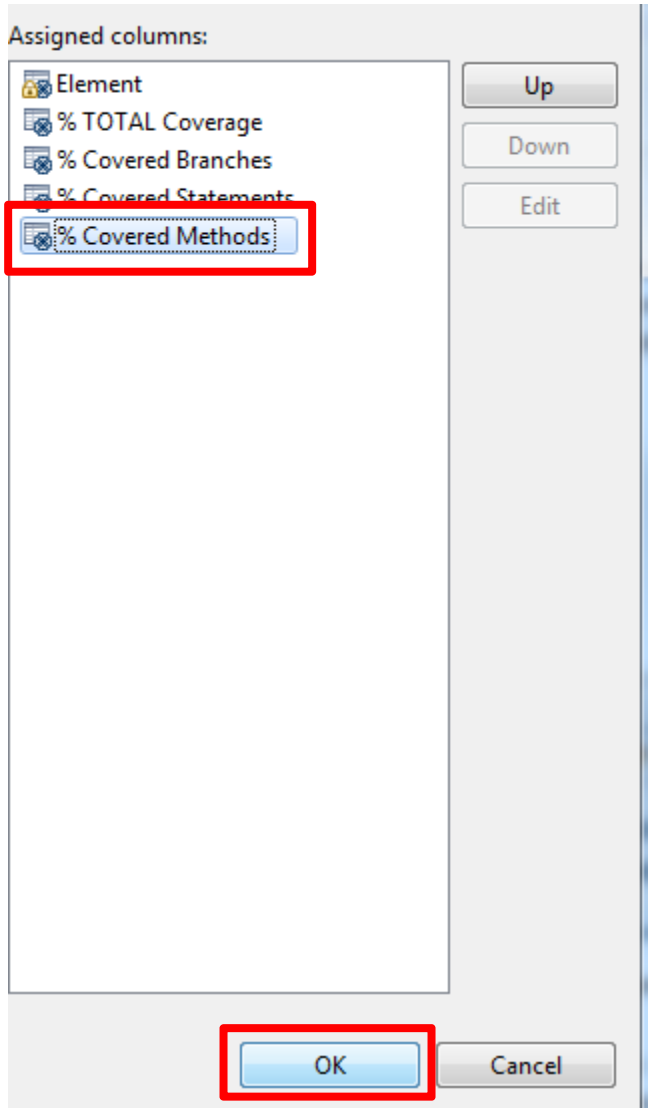
# Selecting %Covered Statements



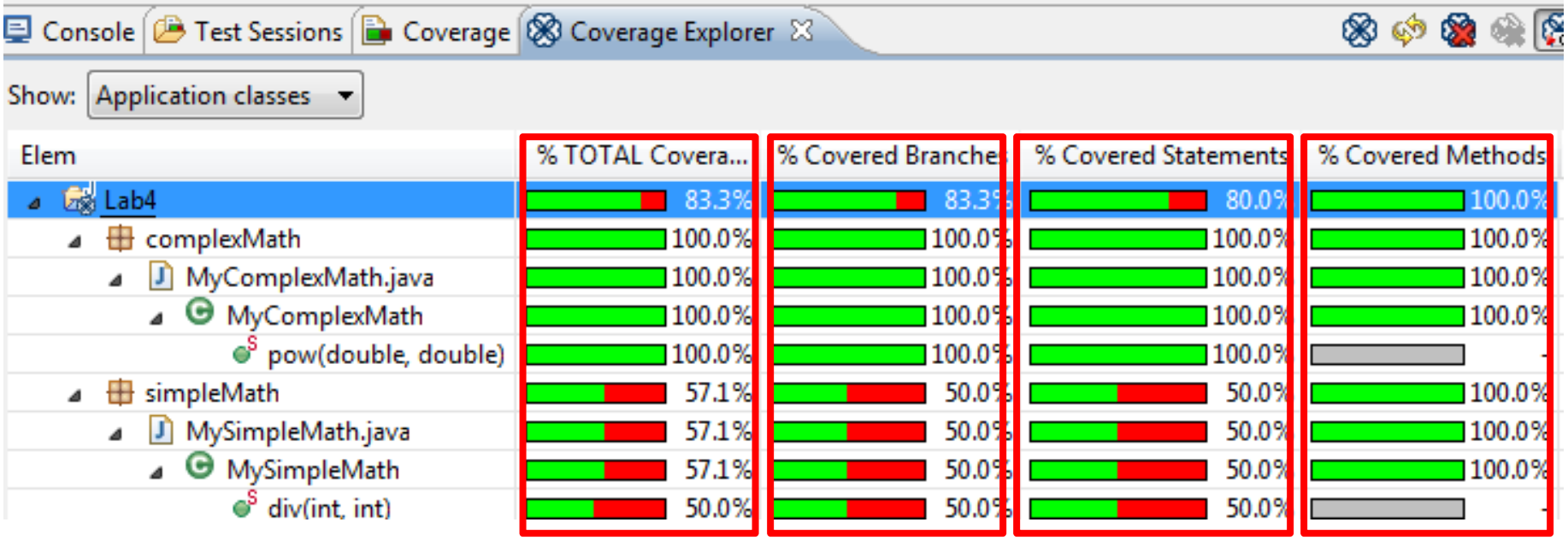
# Selecting %Covered Methods



# Project Code Coverage Analysis

- **Complete Coverage Analysis** of the project, package, file, class and method



# QUIZ - 2

- **Practical Lab Quiz**
  - **Date:** 29, 31 October 2013
  - **Time:** 12:00 to 13:00

- **Topics Covered (slides available on *LMS*)**
  - Junit test method implementation
  - Code Coverage Analysis using Clover

- **Rules**
  - Student will not allow to take a Quiz after 10 minutes of exam session.
  - No excuse for absent student.
  - **In both above cases the student will get 0 (zero) mark.**