



Relational Databases with MySQL Week 12 Coding Assignment

URL to GitHub Repository: https://github.com/quezchristina/mysql_week12

URL to Public Link of your Video: <https://youtu.be/MhLwVP6IPUM>

Instructions :

1. Follow the **Coding Steps** below to complete this assignment.

- In Eclipse, or an IDE of your choice, write the code that accomplishes the objectives listed below. Ensure that the code compiles and runs as directed.
- Create a new repository on GitHub for this week's assignment and push your completed code to this dedicated repo.
- Create a video showcasing your work:
 - In this video: record and present your project verbally while showing the results of the working project.
 - Easy way to Create a video: Start a meeting in Zoom, share your screen, open Eclipse with the code and your Console window, start recording & record yourself describing and running the program showing the results.
 - Your video should be a maximum of 5-minutes.
 - Upload your video with a public link.
 - Easy way to Create a Public Video Link: Upload your video recording to YouTube with a public link.

2. In addition, please include the following in your Coding Assignment Document:

- The URL for this week's GitHub repository.
- The URL of the public link of your video.

3. Save the Coding Assignment Document as a .pdf and do the following:

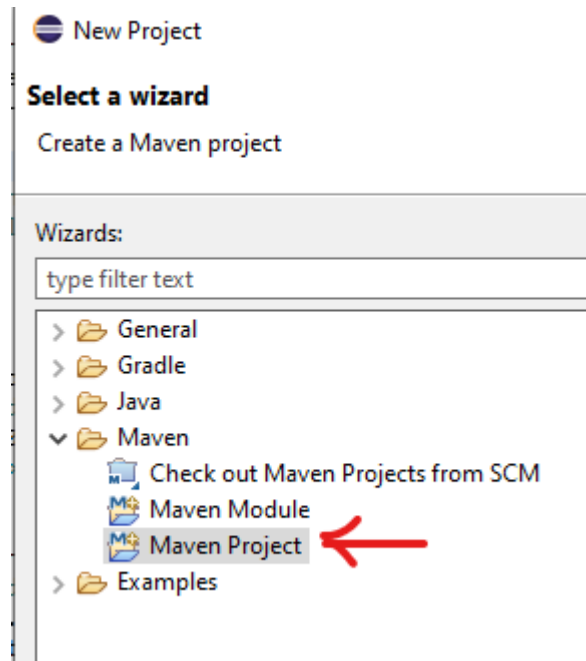
- Push the .pdf to the GitHub repo for this week.
 - Upload the .pdf to the LMS in your Coding Assignment Submission.
-



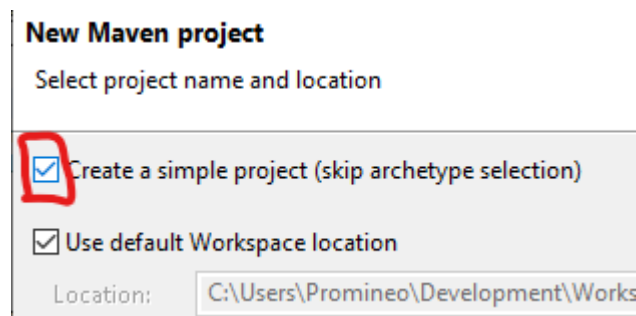
Relational Databases with MySQL Week 12 Coding Assignment

Coding Steps:

1. Create a new Maven project. In Eclipse...
 - a. Right-click in Project Explorer, select "New / Project". Expand "Maven". Select "Maven Project". Click "Next".



- b. Check "Create a simple project (skip archetype selection)". Click "Next".





Relational Databases with MySQL Week 12 Coding Assignment

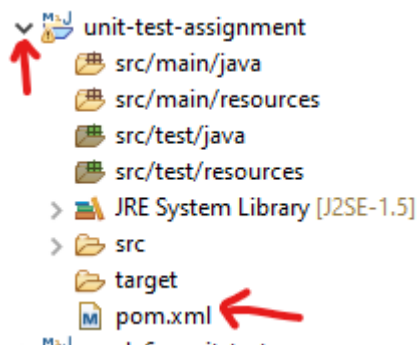
- c. Enter the Group Id: "my.unit.test". Enter the Artifact Id: "unit-test-assignment". Click "Finish".

New Maven project

Configure project

Artifact	
Group Id:	my.unit.test
Artifact Id:	unit-test-assignment
Version:	0.0.1-SNAPSHOT

- d. The project "unit-test-assignment" should appear in the Package Explorer. Click the down arrow next to "unit-test-assignment" to expand it. Double-click on "pom.xml" to open it in the editor.



- e. Put a couple of blank lines between `<version>0.0.1-SNAPSHOT</version>` and `</project>`.

```
1 <project xmlns="http://maven.apache.org/POM/4.0.0" >
2   <modelVersion>4.0.0</modelVersion>
3   <groupId>my.unit.test</groupId>
4   <artifactId>unit-test-assignment</artifactId>
5   <version>0.0.1-SNAPSHOT</version>
6
7   |
8
9 </project>
```



Relational Databases with MySQL Week 12 Coding Assignment

f. Copy and paste the following code into the blank area you just created.

```
<properties>
  <java.version>11</java.version>
  <project.build.sourceEncoding>utf-8</project.build.sourceEncoding>
</properties>

<dependencies>
  <dependency>
    <groupId>com.google.guava</groupId>
    <artifactId>guava</artifactId>
    <version>30.1.1-jre</version>
  </dependency>

  <dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter</artifactId>
    <version>5.7.2</version>
    <scope>test</scope>
  </dependency>

  <dependency>
    <groupId>org.assertj</groupId>
    <artifactId>assertj-core</artifactId>
    <version>3.20.2</version>
    <scope>test</scope>
  </dependency>
```



Relational Databases with MySQL Week 12 Coding Assignment

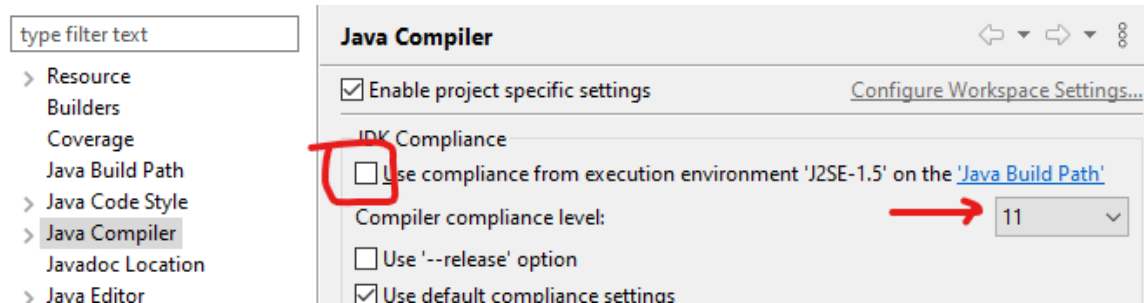
```
<dependency>
  <groupId>org.mockito</groupId>
  <artifactId>mockito-junit-jupiter</artifactId>
  <version>3.11.2</version>
  <scope>test</scope>
</dependency>
</dependencies>

<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.8.1</version>
      <configuration>
        <source>${java.version}</source>
        <target>${java.version}</target>
      </configuration>
    </plugin>
  </plugins>
</build>
```

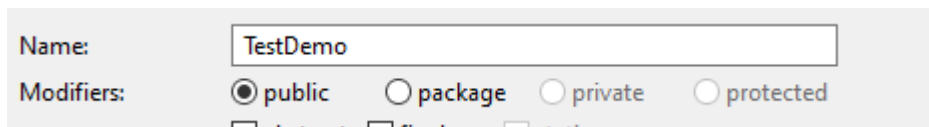
- g. Save the file.
- h. For Eclipse only: right-click on "unit-test-assignment" in the Project Explorer. Click on "Properties". Click "Java Compiler". Make sure "Enable project specific settings" is checked. Uncheck "Use compliance from execution environment 'J2SE-1.5' on the 'Java Build Path'". Set "Compiler compliance level" to 11. Click "Apply and Close".



Relational Databases with MySQL Week 12 Coding Assignment



- i. If asked to rebuild the project, click "Yes".
2. Create a class named "TestDemo" under src/main/java in the default package. (In Package Explorer, expand "unit-test-assignment". Right-click on "src/main/java" and select "New / Class". Enter "TestDemo" in the "Name" field and click "Finish".)



- a. Create an instance method (not static) named addPositive. It should take two int parameters and return an int.

```
public int addPositive(int a, int b) {}
```
 - b. If both parameters are positive (greater than zero) return the sum of the parameters. If either parameter is zero or negative, throw an IllegalArgumentException with the message "Both parameters must be positive!". IllegalArgumentException is in the java.lang package so you won't need an import statement.
 - c. Save the file.
3. In Package Explorer, find "src/test/java" and right-click on it. Select "New / JUnit Test Case". In the "Name" field, enter "TestDemoTest". Make sure that "New JUnit Jupiter test" is selected. Make sure that "@BeforeEach setUp()" is checked. Click "Finish".



Relational Databases with MySQL Week 12 Coding Assignment

☐ New JUnit 3 test ☐ New JUnit 4 test ☒ New JUnit Jupiter test

Source folder:

Package:

Name:

Superclass:

Which method stubs would you like to create?

☐ @BeforeAll setUpBeforeClass() ☐ @AfterAll tearDownAfterClass()

☒ @BeforeEach setUp() ☐ @AfterEach tearDown()

☐ constructor

4. In TestDemoTest.java, add a private instance variable of type TestDemo named testDemo.
 - a. In the setUp method, create the TestDemo object. This will ensure that a new TestDemo object is created before each test.
 - b. Change "@Test" to "@ParameterizedTest". Add the import statement for `org.junit.jupiter.params.ParameterizedTest`.
 - c. Change the name of method "test" to "assertThatTwoPositiveNumbersAreAddedCorrectly".
 - d. Add four parameters to `assertThatTwoPositiveNumbersAreAddedCorrectly` as shown:

Type	Name
int	a
int	b
int	expected
Boolean	expectException

e.



Relational Databases with MySQL Week 12 Coding Assignment

- f. Write the test. Remove the "fail" line. Test the value of `expectException`. If it is false, assert that when `TestDemo.addPositive` is called with values `a` and `b`, that the result is the same as the parameter `expected`. The assertion should look like this:

```
if(!expectException) {  
    assertThat(testDemo.addPositive(a, b)).isEqualTo(expected);  
}
```

- g. Add the test for the thrown exception in an else clause. Use

```
assertThatThrownBy for this. Add the static import  
org.assertj.core.api.Assertions.assertThatThrownBy;
```

- h. As a parameter to `assertThatThrownBy`, add a Lambda expression with no parameters. The Lambda body should be the method call to `testDemo.addPositive`.

- i. Use the assertion `isInstanceOf(IllegalArgumentException.class)` to ensure that the correct exception is thrown.

- j. If this is too confusing, you can "cheat" and copy this:

```
assertThatThrownBy(() ->  
    testDemo.addPositive(a, b))  
    .isInstanceOf(IllegalArgumentException.class);
```

- k. Add the parameter source method.

- i. Create a static method named `argumentsForAddPositive`. It should not have any parameters and it should return a `Stream` of `Arguments`. The imports are: `java.util.stream.Stream` and `org.junit.jupiter.params.provider.Arguments`.

- ii. The method should return a `Stream` as in `Stream.of()`;

- iii. Each parameter set should be wrapped in an `arguments()` method call. Add the static import for `arguments`:
`org.junit.jupiter.params.provider.Arguments.arguments.`



Relational Databases with MySQL Week 12 Coding Assignment

- iv. So, if you are adding 2 and 4 to get the value of 6 and are not expecting an exception, you need to do:

```
arguments(2, 4, 6, false)
```

- v. Add as many arguments lines as needed to test the `addPositive` method thoroughly. Make sure to add some zero or negative arguments.

- 1. Just below the `@ParameterizedTest` annotation, add the annotation `@MethodSource`. Pass a single parameter to `@MethodSource`. It must be the fully-qualified (includes package) class name of the test followed by a `#` sign followed by the name of the method that supplies the parameters. Since the test is in the default package, there is no package in the fully-qualified class name. So,

```
@MethodSource("TestDemoTest#argumentsForAddPositive")
```

- 5. In `TestDemo.java`, add another method named `randomNumberSquared`. This method obtains a random int between 1 and 10 and then returns the square of the number.
 - a. `randomNumberSquared` should return an int and not take any parameters.
 - b. It should call another method in the same class named `getRandomInt`. This method takes no parameters and must be package visibility so that the test can see it. `getRandomInt` should look like this:

```
int getRandomInt() {  
    Random random = new Random();  
    return random.nextInt(10) + 1;  
}
```

The `Random` class is in the `java.util` package.

- c. `randomNumberSquared` should return the value obtained from `getRandomInt` multiplied by itself.
- 6. Write a test for `randomNumberSquared` in `TestDemoTest.java`. Since you don't know what `getRandomInt` will return (that's the point of random, after all), you will need to mock it out and supply a known value.
 - a. Create a method annotated with `@Test` named `assertThatNumberSquaredIsCorrect`. The method must have package visibility (not public!) or JUnit won't find it. The annotation `@Test` is in the `org.junit.jupiter.api` package.



Relational Databases with MySQL Week 12 Coding Assignment

- b. To mock the `TestDemo` class, use `Mockito.spy`. The `spy` method can be imported with a static import of `org.mockito.Mockito.spy`.

```
TestDemo mockDemo = spy(testDemo);
```

- c. Program the mocked `TestDemo` object to return 5 when the `getRandomInt` method is called. Remember to use the form:

```
doReturn(aValue).when(mockedObject).methodCall(); You can use a static
import for doReturn: import static org.mockito.Mockito.doReturn;

doReturn(5).when(mockDemo).getRandomInt();
```

- d. Call the method `randomNumberSquared` on the mocked `TestDemo` object. This will call the stubbed out (mocked) method `getRandomInt`, which now should return the value 5.

```
int fiveSquared = mockDemo.randomNumberSquared();
```

- e. Use `assertThat` to test that the value returned from `randomNumberSquared` is equal to 5 squared.

```
assertThat(fiveSquared).isEqualTo(25);
```

- f. You don't need to verify the mocked method call – you know it was called since the return value is correct.

Screenshots provided below:



Relational Databases with MySQL Week 12 Coding Assignment

The screenshot displays the Eclipse IDE interface with the following components:

- Left Panel:** Contains a project explorer showing a package named `unit-test-assignment` with sub-packages `mysql-java` and `mysql-javaa`.
- Editor:** Displays the `unit-test-assignment/pom.xml` file. The XML content is as follows:

```
1 https://maven.apache.org/xsd/maven-4.0.0.xsd (xsi:schemaLocation)
2 <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/
3   <groupId>my.unit.test</groupId>
4   <artifactId>unit-test-assignment</artifactId>
5   <version>0.0.1-SNAPSHOT</version>
6
7   <properties>
8     <java.version>11</java.version>
9     <project.build.sourceEncoding>utf-8</project.build.sourceEncoding>
10  </properties>
11
12  <dependencies>
13    <dependency>
14      <groupId>com.google.guava</groupId>
15      <artifactId>guava</artifactId>
16      <version>30.1.1-jre</version>
17    </dependency>
18
19    <dependency>
20      <groupId>org.junit.jupiter</groupId>
21      <artifactId>junit-jupiter</artifactId>
22      <version>5.7.2</version>
23      <scope>test</scope>
24    </dependency>
25
26    <dependency>
27      <groupId>org.assertj</groupId>
28      <artifactId>assertj-core</artifactId>
29      <version>3.20.2</version>
30      <scope>test</scope>
31    </dependency>
32
33    <dependency>
34      <groupId>org.mockito</groupId>
35      <artifactId>mockito-junit-jupiter</artifactId>
36      <version>3.11.2</version>
37      <scope>test</scope>
38    </dependency>
39  </dependencies>
40
41  <build>
42    <plugins>
43      <plugin>
44        <groupId>org.apache.maven.plugins</groupId>
45        <artifactId>maven-compiler-plugin</artifactId>
46        <version>3.8.1</version>
47        <configuration>
48          <source>${java.version}</source>
49          <target>${java.version}</target>
50        </configuration>
51      </plugin>
52    </plugins>
53  </build>
54
55 </project>
```
- Bottom Panel:** Includes tabs for `Console`, `Problems`, `Debug Shell`, `History`, `JUnit`, `Git Staging`, and `Coverage`. The `JUnit` tab is active, showing `Runs: 0/0`, `Errors: 0`, and `Failures: 0`.



Relational Databases with MySQL Week 12 Coding Assignment

```
1 import java.util.Random;
2
3 public class TestDemo {
4
5     public int addPositive(int a, int b) {
6         if(a >= 0 && b >= 0) {
7             int sum = a + b;
8             System.out.println(sum);
9             return sum;
10        } else {
11            throw new IllegalArgumentException("Both parameters must be positive!");
12        }
13    }
14
15    public int getRandomInt() {
16        Random random = new Random ();
17        return random.nextInt(10) + 1;
18    }
19
20    //randomNumberSquared
21    // return value obtained from getRandomInt multiple itself
22
23    public int randomNumberSquare() {
24        int randomNumber = getRandomInt();
25        int randomNumberSqaure = randomNumber * randomNumber;
26        return randomNumberSqaure;
27    }
28
29
30
31 }
32
```



Relational Databases with MySQL Week 12 Coding Assignment

The screenshot shows the Eclipse IDE interface. The left sidebar displays the project structure for 'unit-test-assignment', including 'src/main/java' and 'src/test/java'. The main editor window shows the code for 'TestDemoTest.java'. The code includes imports for AssertJ, JUnit, and Mockito, and defines a 'TestDemo' class with methods for adding positive numbers and a static method for adding positive numbers. The test class 'TestDemoTest' contains several test methods, including '@BeforeEach', '@ParameterizedTest', and '@Test' methods that use AssertJ and Mockito for assertions and mocking.

```
1 import static org.assertj.core.api.Assertions.assertThat;
2 import static org.assertj.core.api.Assertions.assertThatThrownBy;
3
4 import java.util.stream.Stream;
5
6 import org.junit.jupiter.api.BeforeEach;
7 import org.junit.jupiter.api.Test;
8 import org.junit.jupiter.params.ParameterizedTest;
9 import org.junit.jupiter.params.provider.Arguments;
10 import org.junit.jupiter.params.provider.MethodSource;
11 import static org.junit.jupiter.params.provider.Arguments.arguments;
12 import static org.mockito.Mockito.doReturn;
13 import static org.mockito.Mockito.spy;
14
15 class TestDemoTest {
16     TestDemo testDemo;
17
18     @BeforeEach
19     void setUp() throws Exception {
20         testDemo = new TestDemo();
21     }
22
23     @ParameterizedTest
24     @MethodSource("TestDemoTestArgumentsForAddPositive")
25     void assertThatTwoPositiveNumbersAreAddedCorrectly(int a, int b, int expected, boolean expectException) {
26         if(!expectException) {
27             assertThat(testDemo.addPositive(a, b)).isEqualTo(expected);
28         } else {
29             assertThatThrownBy(() -> testDemo.addPositive(a, b)).isInstanceOf(IllegalArgumentException.class);
30         }
31     }
32
33     static Stream<Arguments> argumentsForAddPositive(){
34         return Stream.of(Arguments.of(1,2,3,false),
35                         Arguments.of(0,-2,0,true));
36     }
37
38
39
40     @Test
41     void assertThatNumberSquaredIsCorrect() {
42         TestDemo mockDemo = spy(testDemo);
43         doReturn(5).when(mockDemo).getRandomInt();
44
45         int fiveSquared = mockDemo.randomNumberSquare();
46         assertThat(fiveSquared).isEqualTo(25);
47     }
48
49
50
51
52
53
54
55
```



PROMINEO TECH

Relational Databases with MySQL Week 12 Coding Assignment

The screenshot shows the Eclipse IDE interface. The top toolbar includes icons for Debug, Project Explorer, Package Explorer, and Run. The Package Explorer on the left shows the project structure: 'unit-test-assignment' (unit-test-assignment) with sub-packages 'src/main/java' and 'src/test/java'. The 'src/test/java' package contains 'TestDemoTest.java'. The main editor displays the code for 'TestDemoTest.java', which includes imports for JUnit 5 and Mockito, and a test class 'TestDemoTest' with methods like 'setUp()', 'addPositive()', and 'square()'. The bottom console shows the test results: 'Finished after 1 seconds', 'Runs: 3/3', 'Errors: 0', and 'Failures: 0'. The test runner is 'JUnit 5' and the test class is 'TestDemoTest'.