

CSSE2002/7023  
IntelliJ JUnit Guide  
Version: 1.0

Written by: Course Staff

2019

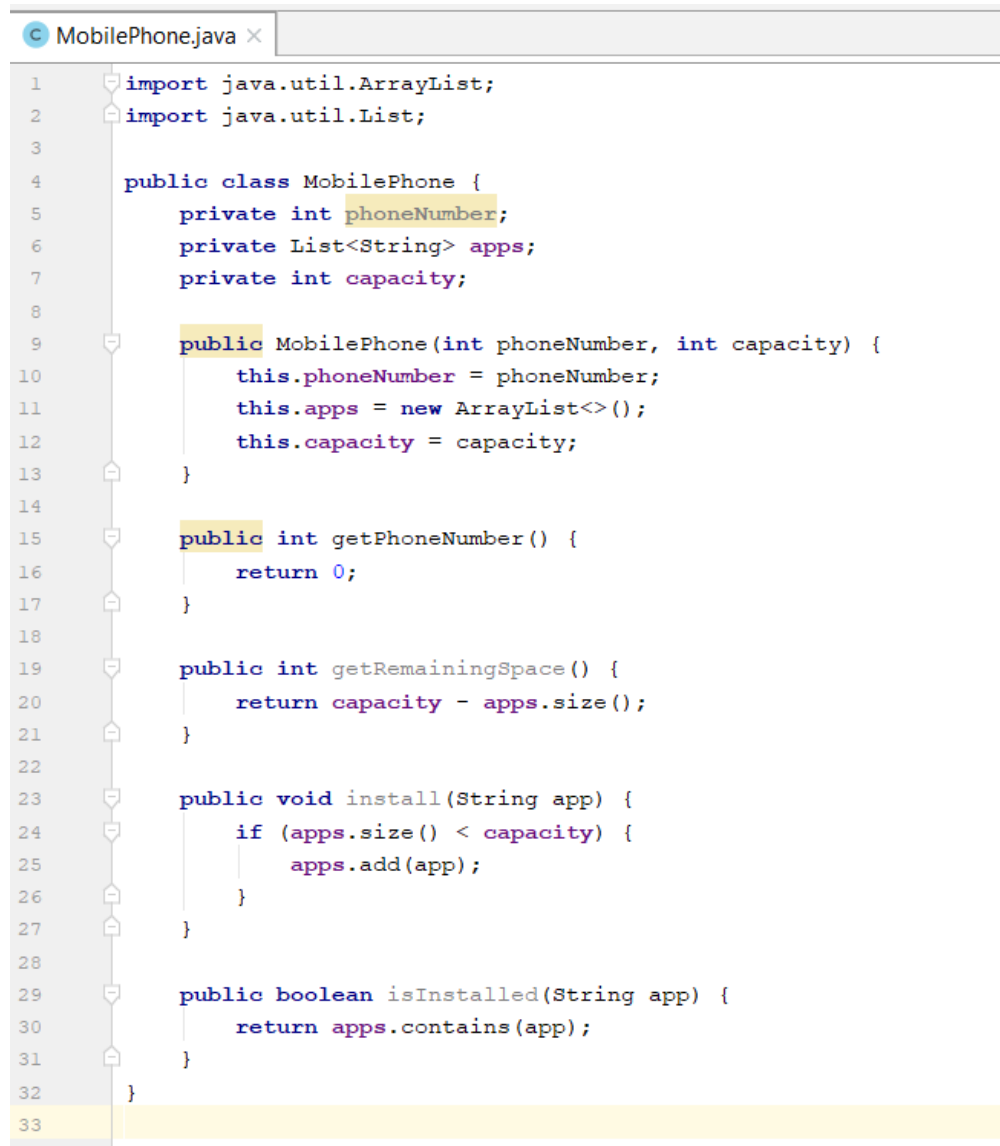
## 1 Testing Code

Testing your code is a vitally important part of writing software — just because your code compiles, or doesn't crash, doesn't necessarily mean that it is working as expected, especially in edge cases. Thoroughly testing your code using a testing framework allows you to have more confidence that your code is working correctly.

In this course, we will be using JUnit 4, a unit testing framework for Java code. This guide will demonstrate how to go about setting up a JUnit test suite in IntelliJ IDEA.

## 2 Setting Up

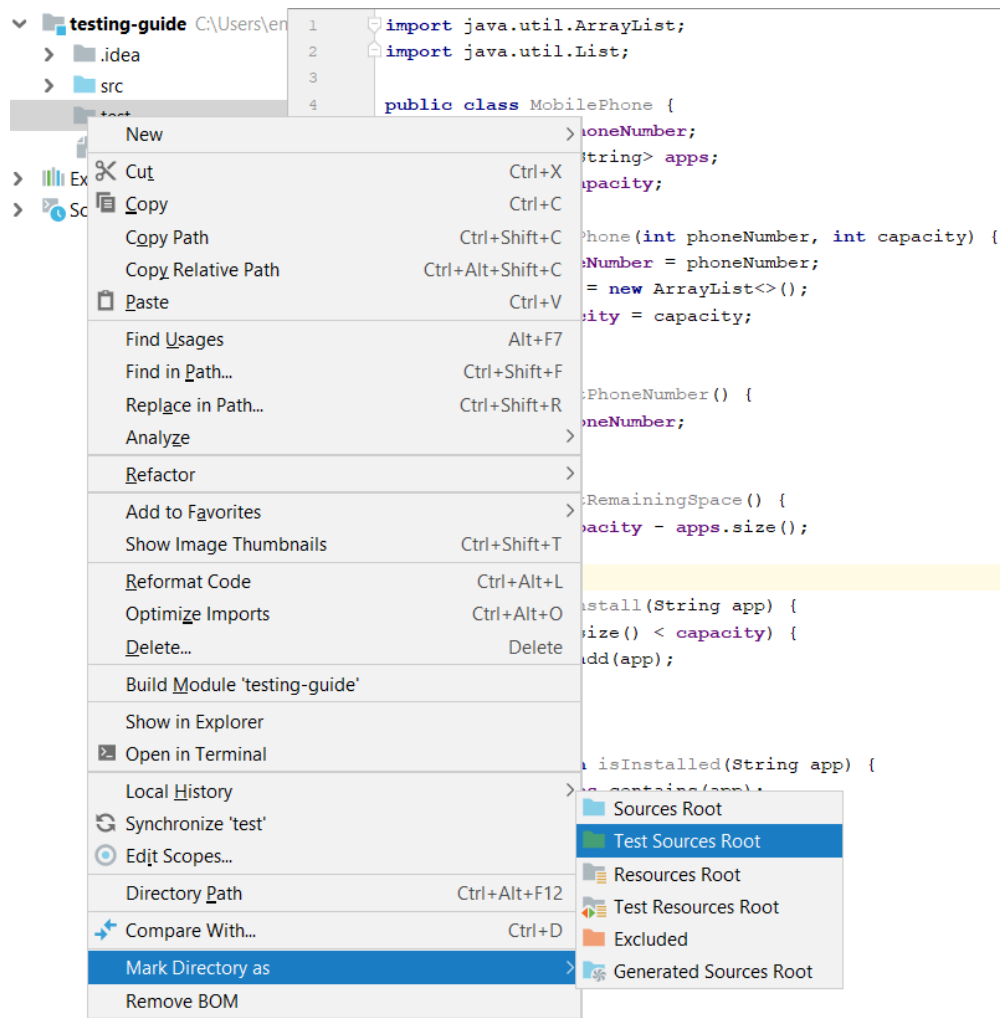
In order to create a test suite, we should first write some code to test. Create a new project in IntelliJ, and create the following class:



```
1  import java.util.ArrayList;
2  import java.util.List;
3
4  public class MobilePhone {
5      private int phoneNumber;
6      private List<String> apps;
7      private int capacity;
8
9      public MobilePhone(int phoneNumber, int capacity) {
10         this.phoneNumber = phoneNumber;
11         this.apps = new ArrayList<>();
12         this.capacity = capacity;
13     }
14
15     public int getPhoneNumber() {
16         return 0;
17     }
18
19     public int getRemainingSpace() {
20         return capacity - apps.size();
21     }
22
23     public void install(String app) {
24         if (apps.size() < capacity) {
25             apps.add(app);
26         }
27     }
28
29     public boolean isInstalled(String app) {
30         return apps.contains(app);
31     }
32 }
33
```

### 3 Creating Test Folder

As we know, in IntelliJ, the code source root is called `src` and is coloured blue to indicate that `.java` files can be found there. Tests work in a similar manner — right click on your project name in the navigation panel on the left hand side and select `New > Directory`. When prompted, name this directory `test`. After creating this directory, right click on it and select `Mark Directory as > Test Sources Root`.



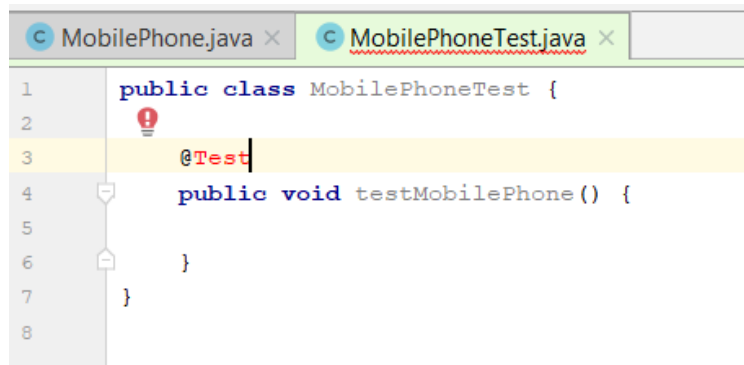
This should change the colour of the directory to green, indicating that test code can be found here.

## 4 Creating JUnit Tests

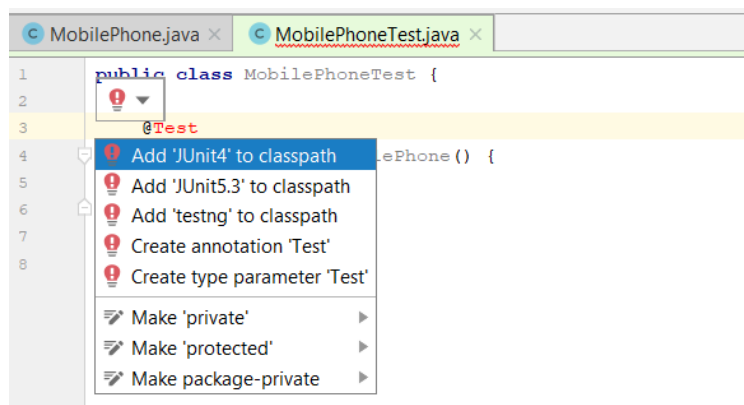
There are many ways to go about creating a new test suite, two of which are outlined below.

### 4.1 Manually Creating Classes

JUnit tests are just ordinary Java classes, so one method of creating a test for our `MobilePhone` class is to create a new class (we'll call this `MobilePhoneTest`) in the new `test` folder. Creating a basic test method, as demonstrated in lectures, will result in a `Cannot resolve symbol 'Test'` compilation error to occur. This is because JUnit 4 is not currently in our classpath, meaning that IntelliJ effectively does not know it exists. Clicking on the `@Test` keyword should cause a red lightbulb to appear, as shown below.



Clicking the lightbulb should cause the prompt shown below to appear, and clicking the Add 'JUnit4' to classpath option should make your code compile. **Ensure that you select the correct JUnit version.**

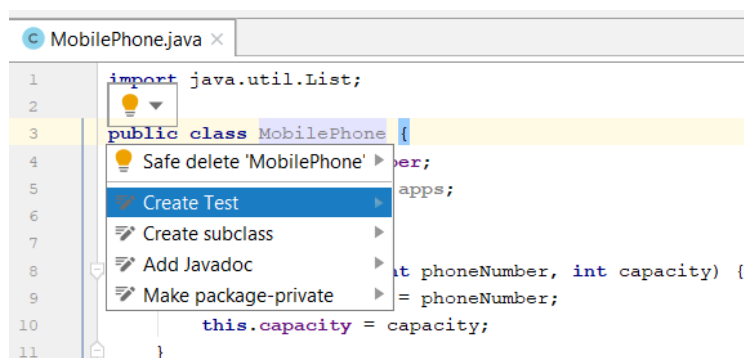


A full test suite can now be written as shown in lectures.

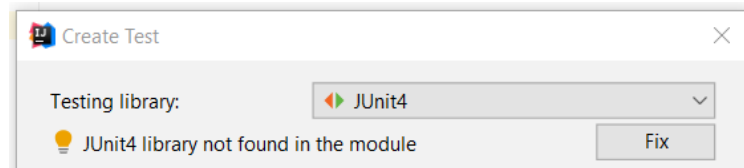
## 4.2 IntelliJ Auto-Generated Tests

In addition to the method outlined above, IntelliJ offers a streamlined, simplified version of this process. **Remember, however, that the ability to write JUnit tests, including the correct method signatures, is examinable content, so do not rely solely on IntelliJ to do this for you.**

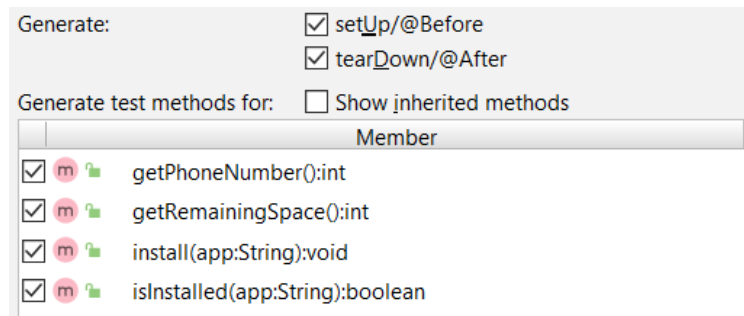
To auto-generate test stubs, click on the name of the class you want to create a test suite for (in this case, `MobilePhone`), and a yellow lightbulb should appear. Clicking on this lightbulb should give the option to **Create Test**.



Clicking this option should cause a new window to pop up. Ensure that the testing library is set to normal JUnit4 from the dropdown menu. A lightbulb may appear saying ‘JUnit4 library not found in the module’.



This window also gives the options for creating `@Before` and `@After` methods, as well as all of the currently existing methods in the class you are creating the test for. Select the methods you would like to add to your class, and then press OK.



This will generate the selected test stubs for you, and should result in JUnit 4 being on your classpath.

## 5 Running Tests

As with running code, the simplest way to run the newly created test suite is to press the play buttons in the gutter.

```

1  import ...
6
7  public class MobilePhoneTest {
8
9      @Before
10     public void setUp() throws Exception {
11     }
12
13     @After
14     public void tearDown() throws Exception {
15     }
16
17     @Test
18     public void getPhoneNumber() {
19     }
20
21     @Test
22     public void getRemainingSpace() {
23     }
24
25     @Test
26     public void install() {
27     }
28
29     @Test
30     public void isInstalled() {
31     }
32 }
33

```

The double play button will run all of the tests, while each single play button will run its corresponding test. Running the tests should cause the testing console to appear at the bottom of the screen. All of the tests should be passing, as we have not yet written any testing code.

✓ MobilePhoneTest	7 ms
✓ getPhoneNumber	4 ms
✓ getRemainingSpace	2 ms
✓ isInstalled	1 ms
✓ install	0 ms

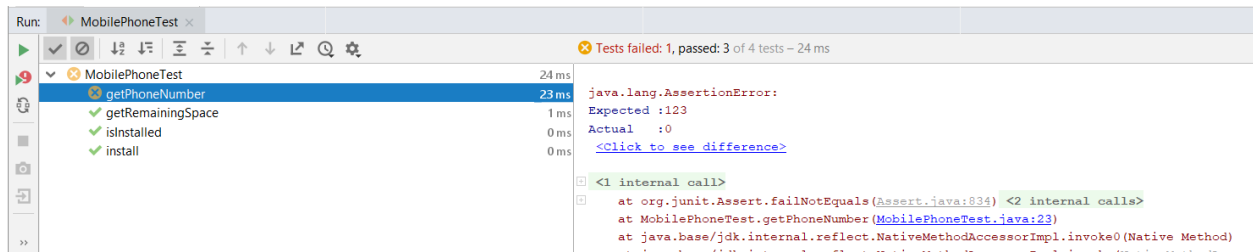
Add the following code to your test class:

```

8  public class MobilePhoneTest {
9      private MobilePhone phone;
10
11     @Before
12     public void setUp() throws Exception {
13         phone = new MobilePhone( phoneNumber: 123, capacity: 3);
14     }
15
16     @After
17     public void tearDown() throws Exception {...}
18
19
20
21     @Test
22     public void getPhoneNumber() {
23         Assert.assertEquals( expected: 123, phone.getPhoneNumber());
24     }
25

```

Re-running the tests should cause one of the tests to fail, namely `getPhoneNumber`.



The console will not only tell you which tests you've failed, but also what line you failed on (in the stack trace), as well as the expected and actual values were for the failed test (in this case, we were expecting 123 to be the phone number, but the actual result returned was 0).

Clicking on the blue highlighted `MobilePhoneTest.java:23` should take us to the line of the test that failed, which in this case is line 23. If we check the `MobilePhone.getPhoneNumber` method, we see that it is returning 0 instead of the stored phone number. If we change this like to be `return phoneNumber;` instead, and then rerun the tests, we will see that all of the tests are now passing.

Note that a test failing could mean that either your test is broken, or your code is broken, and you should consider both of these as possibilities.