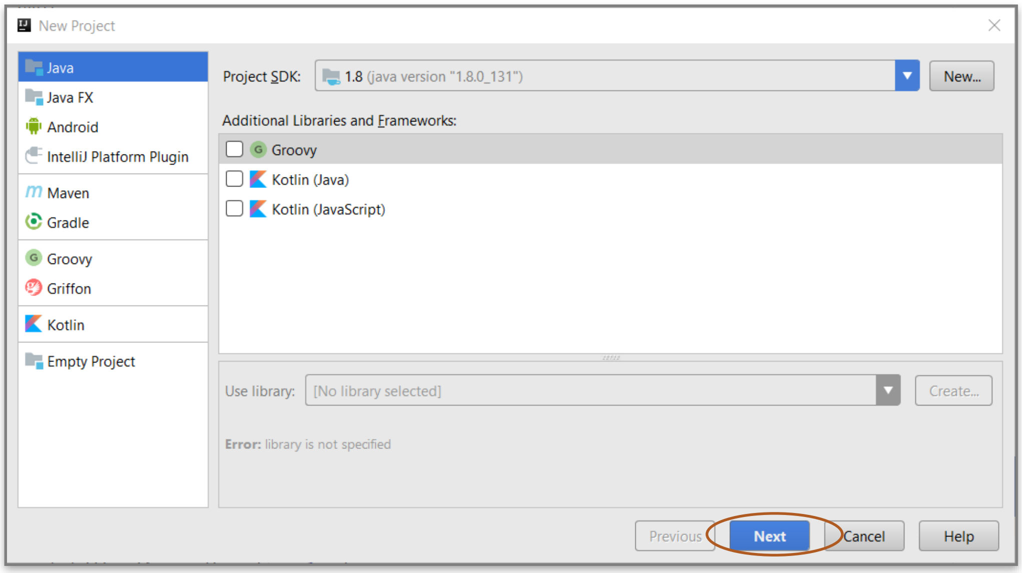
IntelliJ IDEA Tutorial

Contents

Creating New Java Project 2
Opening A Project 4
Creating New Java Class 5
Running Your Application 6
Debugging Your Application 10
Adding Libraries 13
Testing Your Code 20
Running The Inspection Tool 30

Creating New Java Project

Click on **File -> New -> Project…** to open the **New Project** dialog box.



New Project

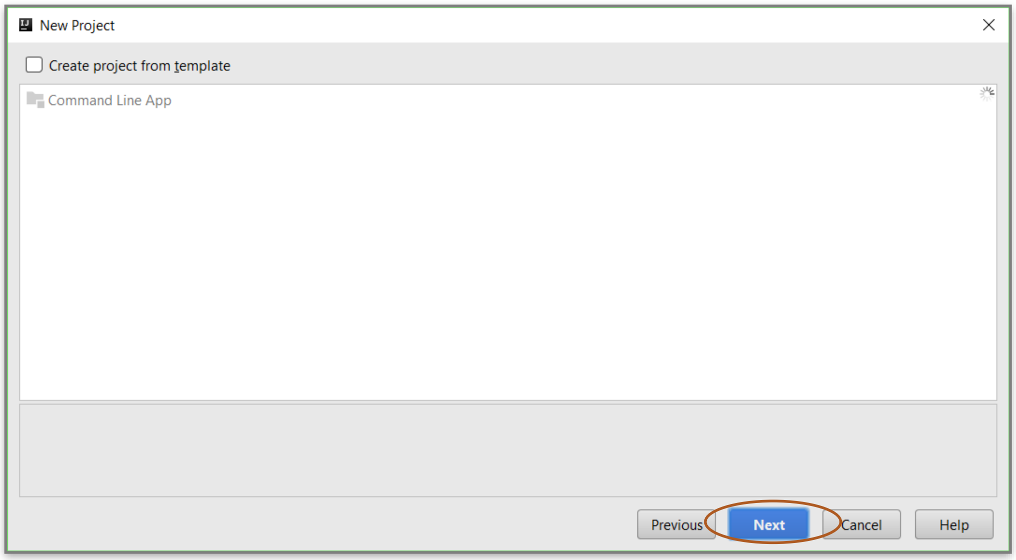
Project SDK: 1.8 (java version "1.8.0_131")

Additional Libraries and Frameworks:

Use library: [No library selected]

Error: library is not specified

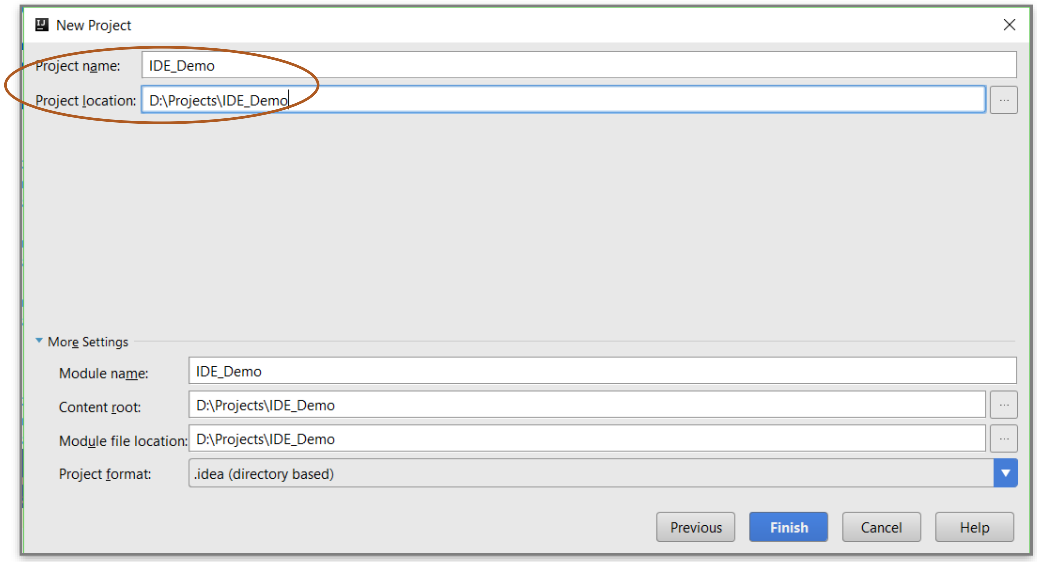
Click on **Next**



New Project

Create project from template

Pick a name for the project and select a directory for storing the project files, then click on **Finish**.


New Project

Project name: IDE_Demo

Project location: D:\Projects\IDE_Demo

More Settings

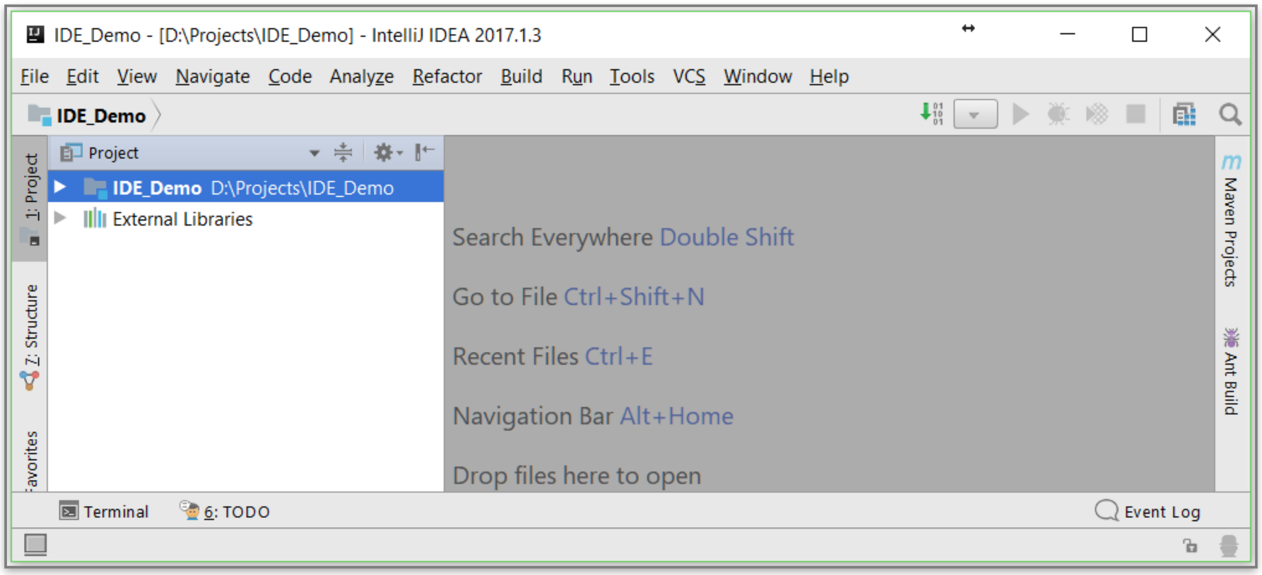
Module name: IDE_Demo

Content root: D:\Projects\IDE_Demo

Module file location: D:\Projects\IDE_Demo

Project format: .idea (directory based)

A new Java project will be created for you and opened in the IDE.


IDE_Demo - [D:\Projects\IDE_Demo] - IntelliJ IDEA 2017.1.3

IDE_Demo D:\Projects\IDE_Demo

External Libraries

Search Everywhere Double Shift

Go to File Ctrl+Shift+N

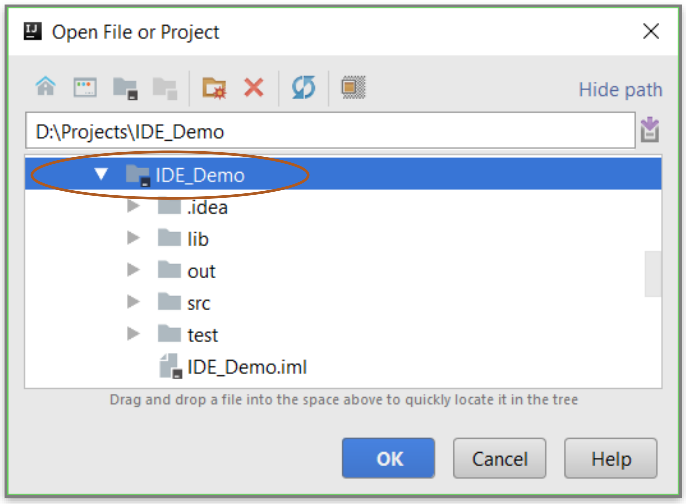
Recent Files Ctrl+E

Navigation Bar Alt+Home

Drop files here to open

Opening A Project

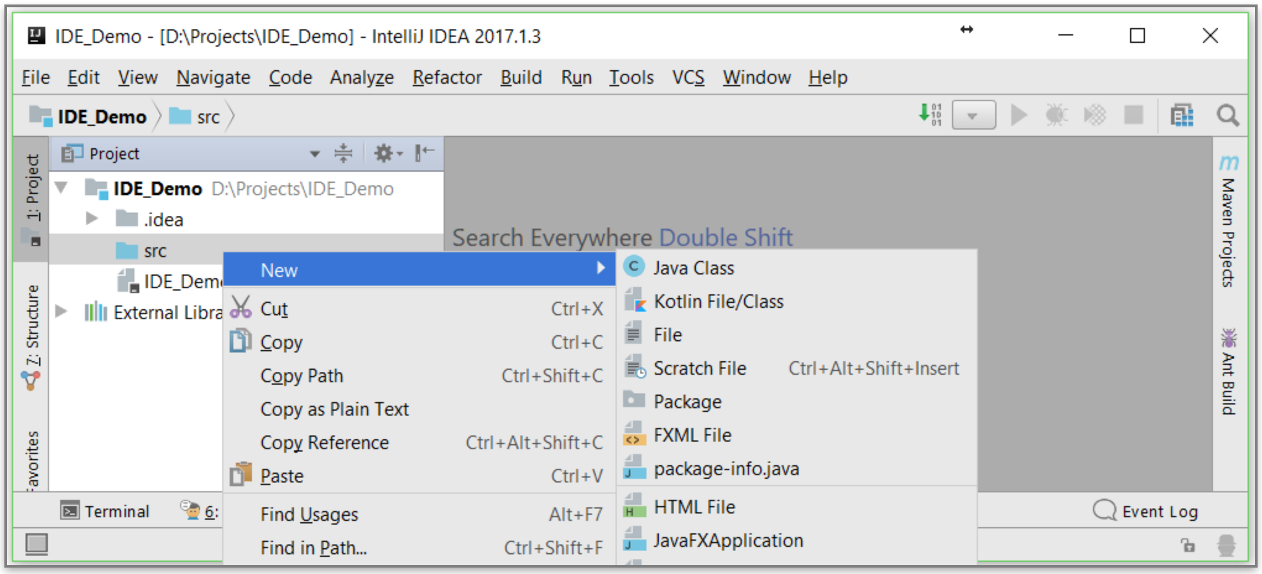
To open a previously saved IntelliJ IDEA Java project, click on **File -> Open…** Browse to and select ***the root folder of the project*** (i.e., the folder containing the .iml file of the project) you want to open, then click on **OK**.



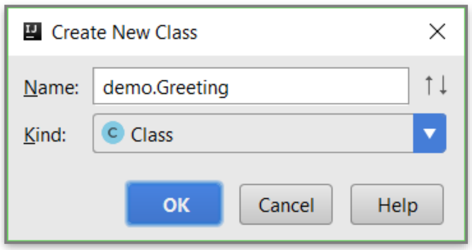
Note: Clicking on **OK** when the project file (with file extension name **.iml**) is selected will open the project file itself, not the whole project.

Creating New Java Class

Right click on folder **src** of your project, then click on **New -> Java Class**.



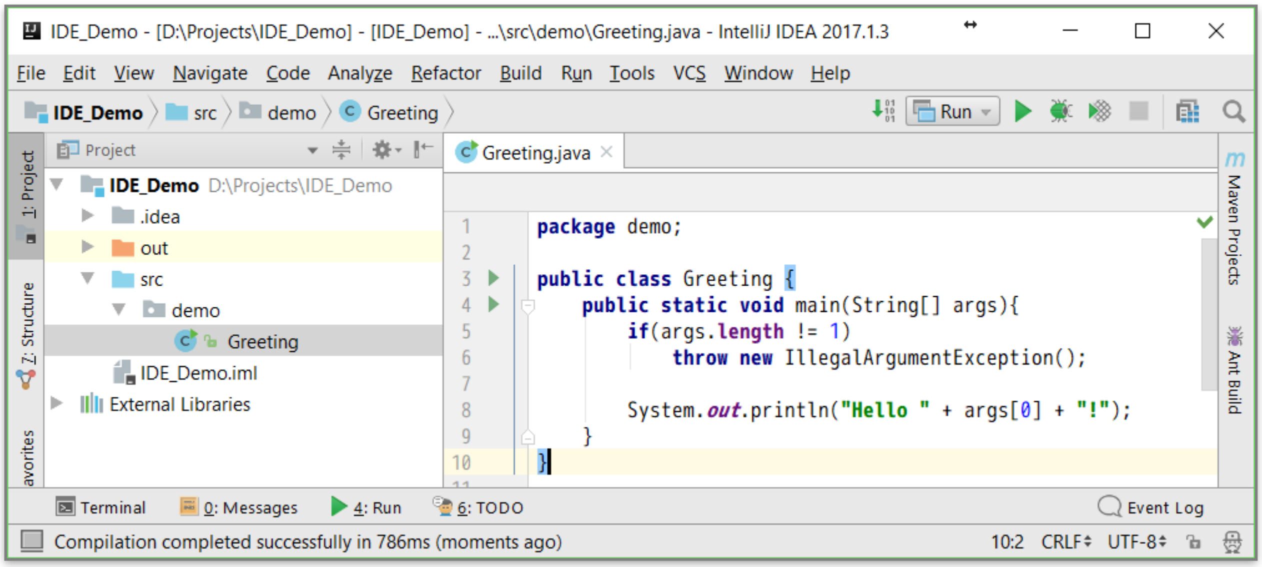
Give the fully qualified name of your class. According to Java naming convention, package names should be in lower case and class names in upper case.



Name: demo.Greeting

Kind: Class

After clicking on **OK**, the Java class and corresponding package structure will be created for you. Double click on the class name to open the source file of the class in the Editor. Here our program expects exactly one argument and will print some greeting message.

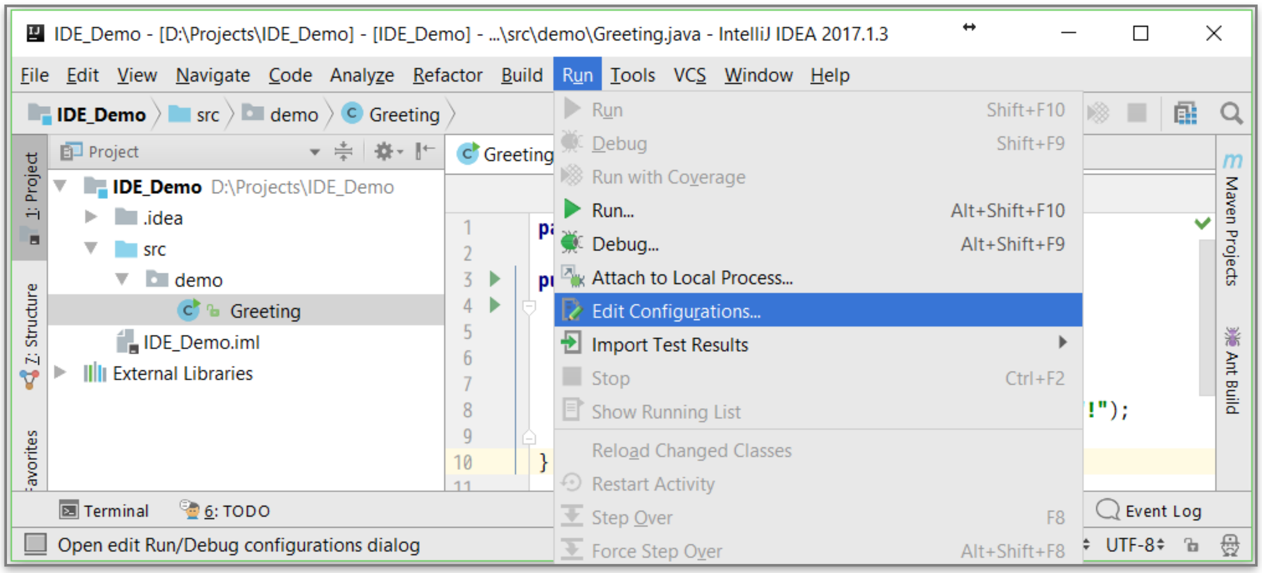

IDE_Demo - [D:\Projects\IDE_Demo] - [IDE_Demo] - ...\src\demo\Greeting.java - IntelliJ IDEA 2017.1.3
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
IDE_Demo src demo Greeting
Project
IDE_Demo D:\Projects\IDE_Demo
.idea
out
src
demo
Greeting
IDE_Demo.iml
External Libraries
Greeting.java
Terminal
0: Messages
4: Run
6: TODO
Event Log

```java
package demo;

public class Greeting {
    public static void main(String[] args){
        if(args.length != 1)
            throw new IllegalArgumentException();

        System.out.println("Hello " + args[0] + "!");
    }
}
```

Compilation completed successfully in 786ms (moments ago)

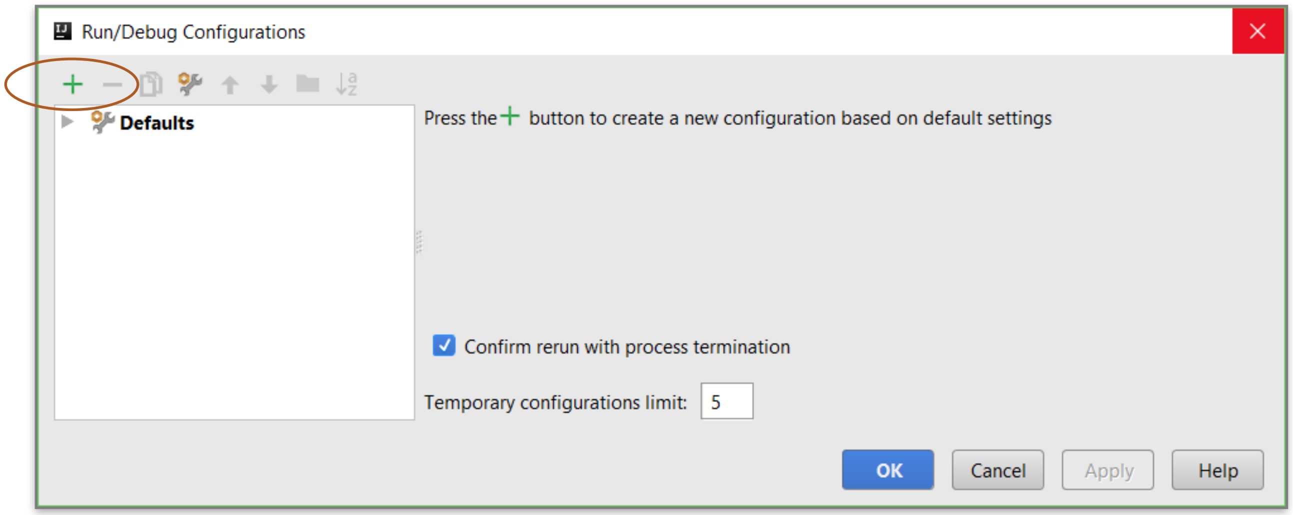
Running Your Application

Before you can run the application, you first need to have the right configuration to be used for launching the application. To create a configuration, click on **Run -> Edit Configurations…**


IDE_Demo - [D:\Projects\IDE_Demo] - [IDE_Demo] - ...\src\demo\Greeting.java - IntelliJ IDEA 2017.1.3
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
Run Shift+F10
Debug Shift+F9
Run with Coverage
Run... Alt+Shift+F10
Debug... Alt+Shift+F9
Attach to Local Process...
Edit Configurations...
Import Test Results
Stop Ctrl+F2
Show Running List
Reload Changed Classes
Restart Activity
Step Over F8
Force Step Over Alt+Shift+F8
Terminal
6: TODO
Event Log

Open edit Run/Debug configurations dialog

The **Run/Debug Configurations** dialog will open.

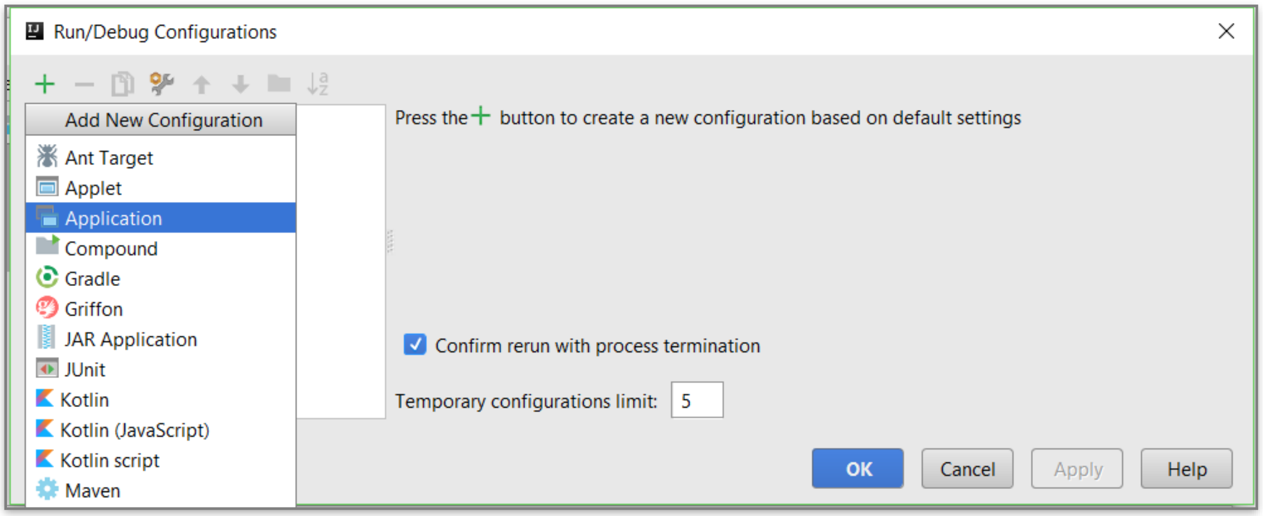


Press the + button to create a new configuration based on default settings

☑ Confirm rerun with process termination

Temporary configurations limit: 5

Click on the **+** button and then select the type of configuration you want to add. Here we want to run the program as an Java application, so we choose **Application** from the list.

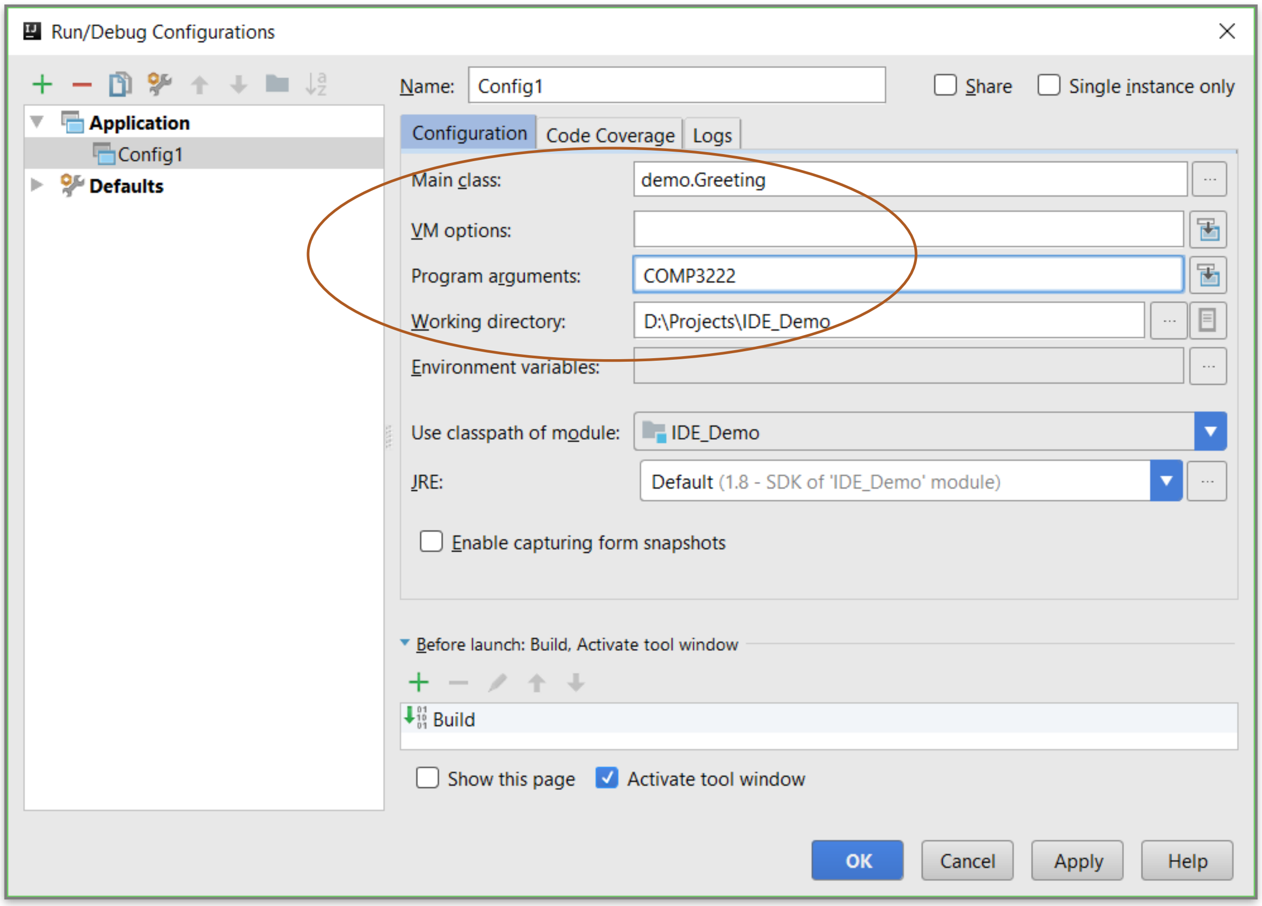


Press the + button to create a new configuration based on default settings

☑ Confirm rerun with process termination

Temporary configurations limit: 5

Name your configuration, input the **Main class** for the application (execution will start from the main method of the **Main class**), and provide all the **Program arguments**, if needed.


Run/Debug Configurations

Application
Config1
Defaults

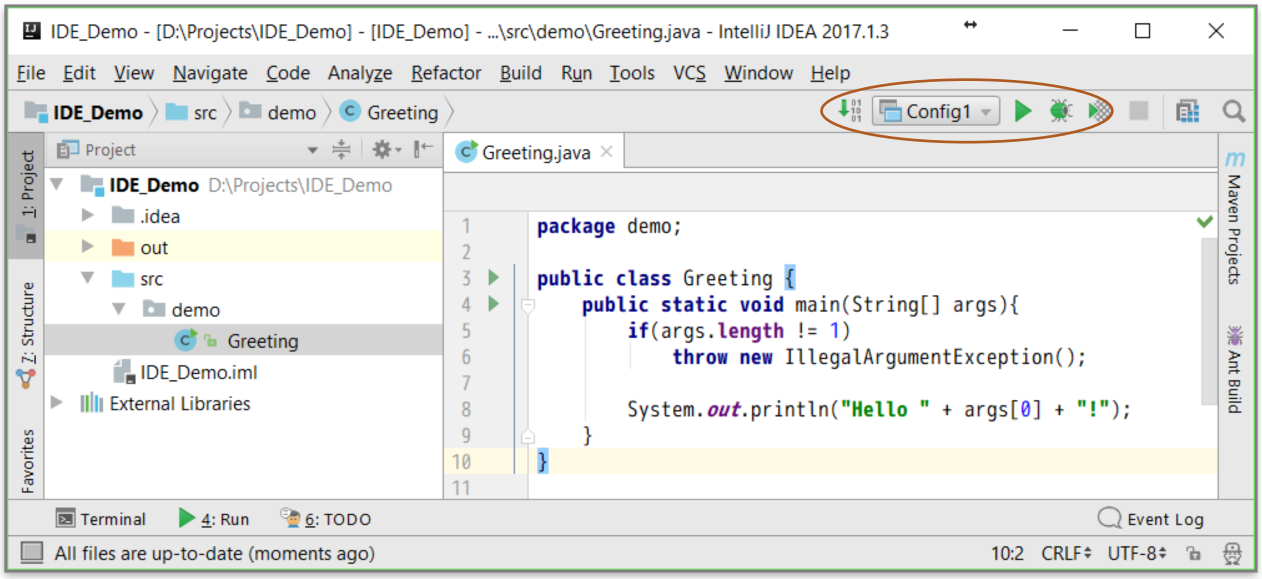
Name: Config1
Share [ ]
Single instance only [ ]

Main class: demo.Greeting
VM options:
Program arguments: COMP3222
Working directory: D:\Projects\IDE_Demo
Environment variables:
Use classpath of module: IDE_Demo
JRE: Default (1.8 - SDK of 'IDE_Demo' module)
[ ] Enable capturing form snapshots

Before launch: Build, Activate tool window
Build

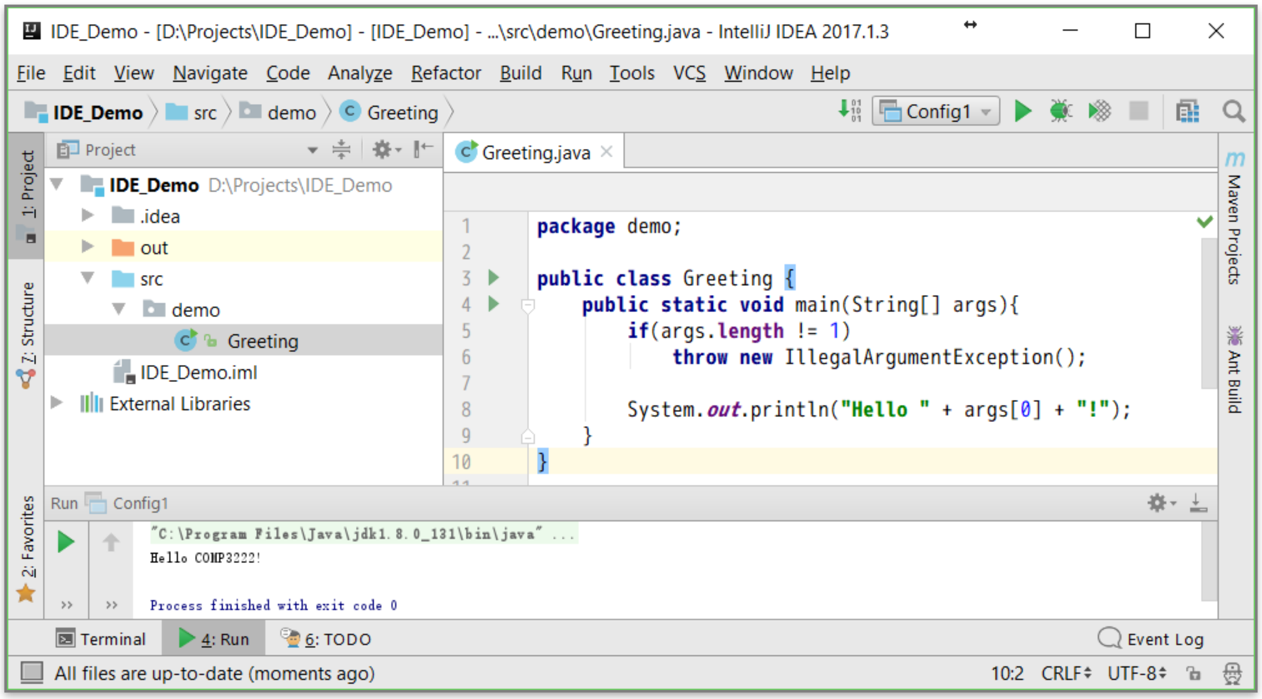
[ ] Show this page
[✓] Activate tool window

After clicking on **OK**, you will see the configuration appears in the toolbar. Next to it are the buttons for running/debugging the application.


IDE_Demo - [D:\Projects\IDE_Demo] - [IDE_Demo] - ...\src\demo\Greeting.java - IntelliJ IDEA 2017.1.3
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
IDE_Demo src demo Greeting
Config1
Project
IDE_Demo D:\Projects\IDE_Demo
.idea
out
src
demo
Greeting
IDE_Demo.iml
External Libraries
Greeting.java
Terminal
4: Run
6: TODO
Event Log
All files are up-to-date (moments ago)
10:2 CRLF UTF-8

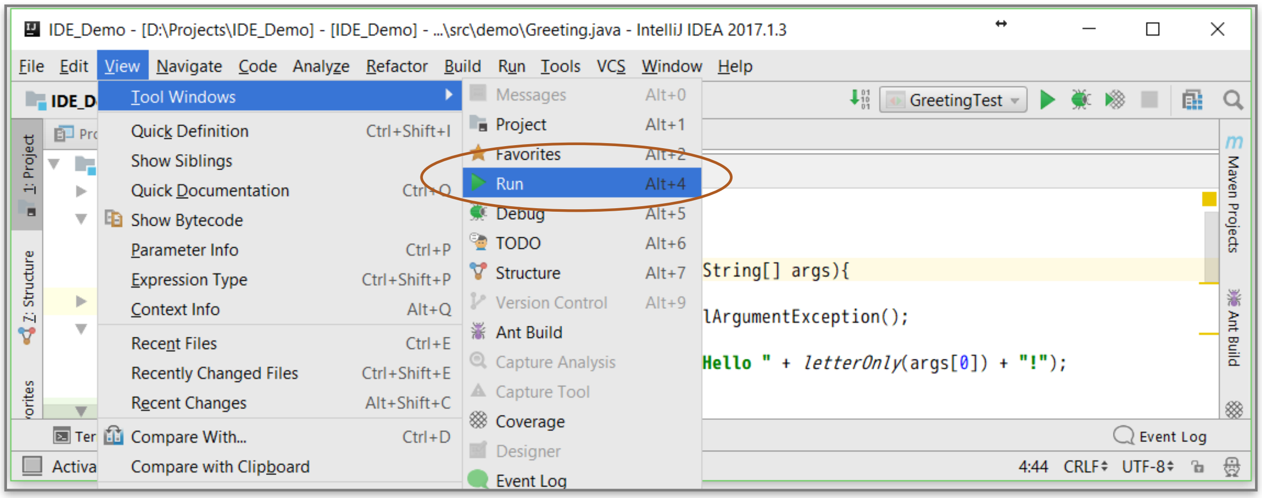
```java
package demo;

public class Greeting {
    public static void main(String[] args){
        if(args.length != 1)
            throw new IllegalArgumentException();

        System.out.println("Hello " + args[0] + "!");
    }
}
```

Click on the **Run** button to launch the application. The output will be shown in the **Run** tool window.



```
"C:\Program Files\Java\jdk1.8.0_131\bin\java" ...
Hello COMP3222!

Process finished with exit code 0
```

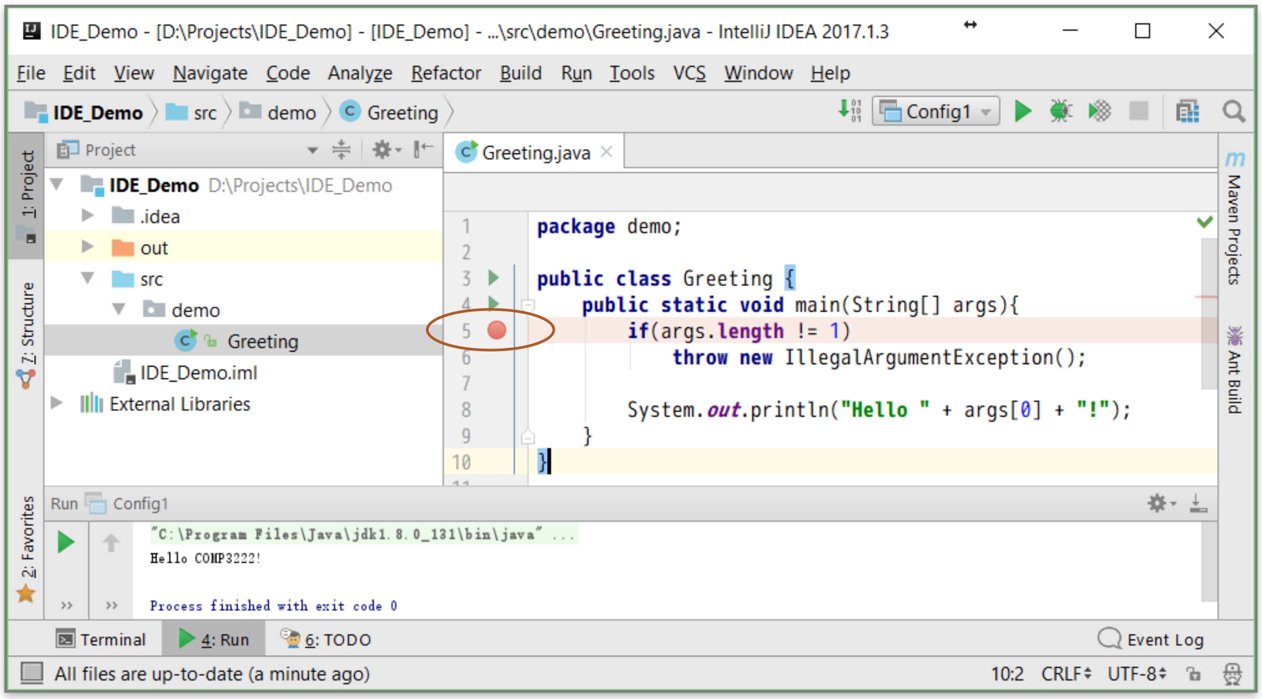
If the tool window is not opened automatically, you can manually open it by clicking on **View -> Tool Windows -> Run**.



```
String[] args){
lArgumentException();
Hello " + letterOnly(args[0]) + "!");
```

Debugging Your Application

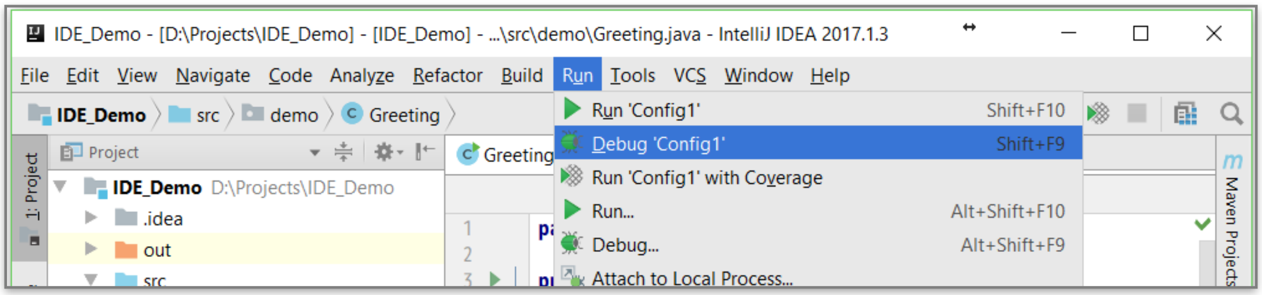
You can click before an executable line to toggle the breakpoint on that line.

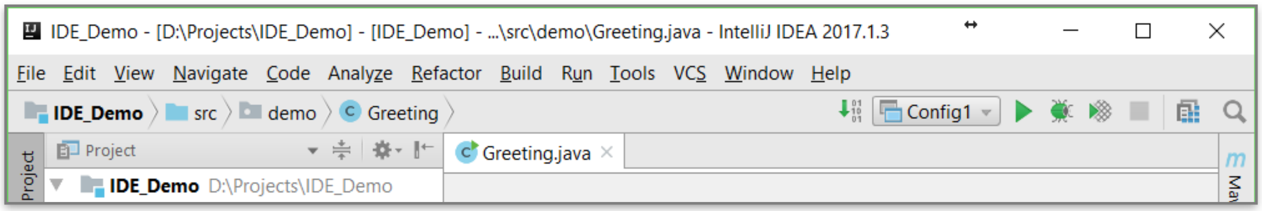

IDE_Demo - [D:\Projects\IDE_Demo] - [IDE_Demo] - ...\src\demo\Greeting.java - IntelliJ IDEA 2017.1.3
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
IDE_Demo src demo Greeting
Config1
Project
IDE_Demo D:\Projects\IDE_Demo
.idea
out
src
demo
Greeting
IDE_Demo.iml
External Libraries
Greeting.java
Run Config1
Terminal
4: Run
6: TODO
Event Log
All files are up-to-date (a minute ago)
10:2 CRLF UTF-8

```
package demo;

public class Greeting {
    public static void main(String[] args){
        if(args.length != 1)
            throw new IllegalArgumentException();

        System.out.println("Hello " + args[0] + "!");
    }
}
```

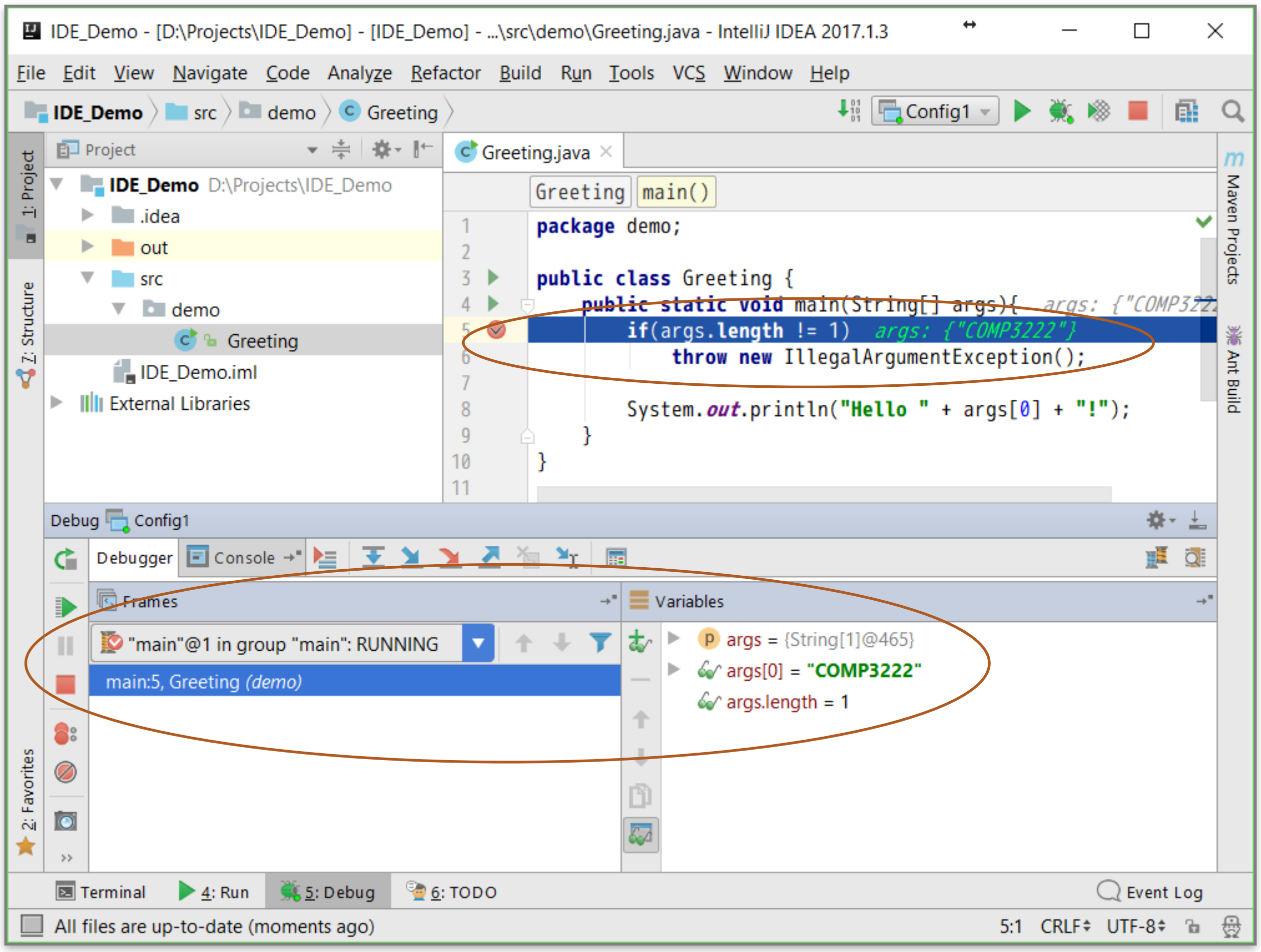
```
"C:\Program Files\Java\jdk1.8.0_131\bin\java" ...
Hello COMP3222!

Process finished with exit code 0
```

Start debugging by clicking on **Run -> Debug 'Config1'** or simply the **Debug** tool button.


IDE_Demo - [D:\Projects\IDE_Demo] - [IDE_Demo] - ...\src\demo\Greeting.java - IntelliJ IDEA 2017.1.3
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
Run 'Config1' Shift+F10
Debug 'Config1' Shift+F9
Run 'Config1' with Coverage
Run... Alt+Shift+F10
Debug... Alt+Shift+F9
Attach to Local Process...



During debugging, the program execution will stop before each line with an active breakpoint, allowing you to inspect the values of variables before executing that line.



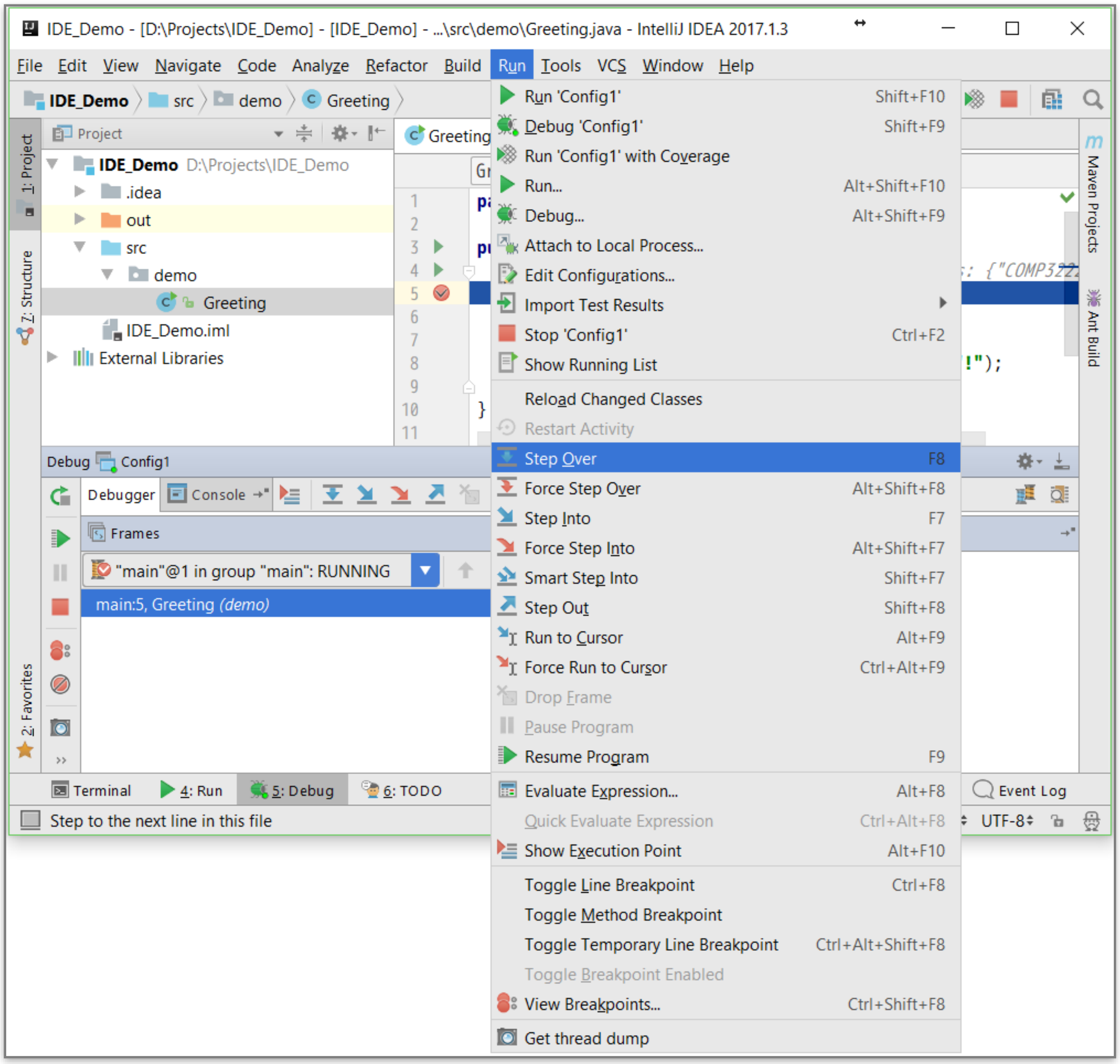
```
package demo;

public class Greeting {
    public static void main(String[] args){   args: {"COMP322
        if(args.length != 1)   args: {"COMP3222"}
            throw new IllegalArgumentException();

        System.out.println("Hello " + args[0] + "!");
    }
}
```

```
"main"@1 in group "main": RUNNING
main:5, Greeting (demo)
```

```
args = {String[1]@465}
args[0] = "COMP3222"
args.length = 1
```

All files are up-to-date (moments ago)

When the execution is stopped during debugging, you may also opt for executing the next statement in a single step (**Step Over**), executing the next statement in a single step unless it involves a method call, in which case the execution should follow the method call into the callee method (**Step Into**), or continuing the execution until the current method has returned to its caller (**Step Out**).

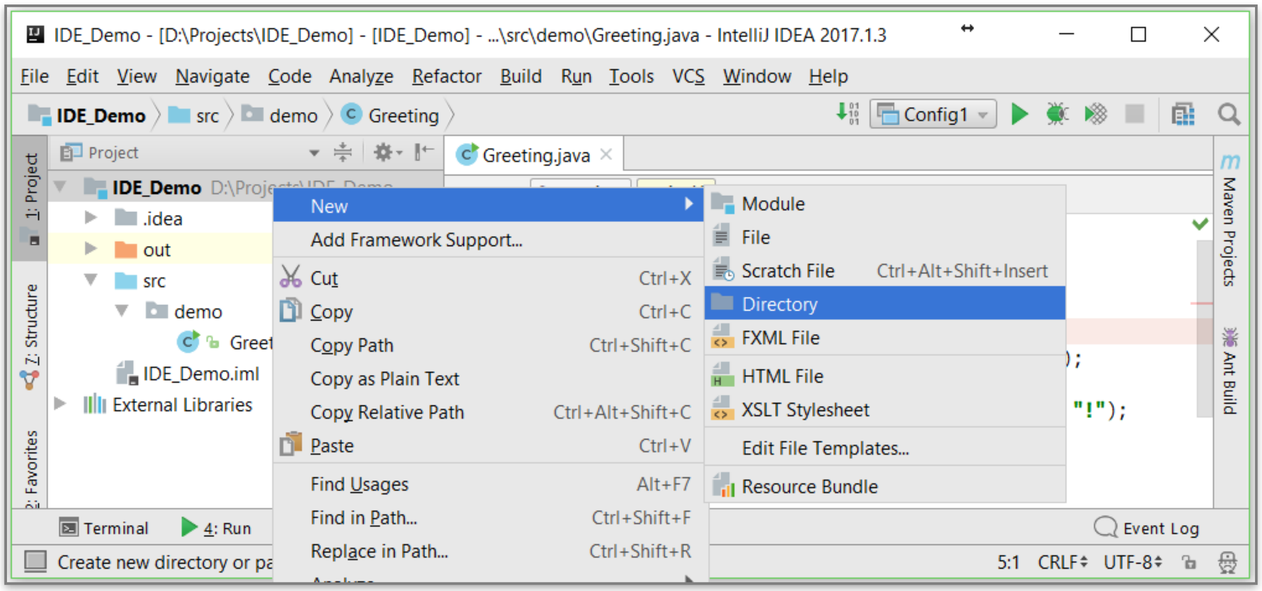

IDE_Demo - [D:\Projects\IDE_Demo] - [IDE_Demo] - ...\src\demo\Greeting.java - IntelliJ IDEA 2017.1.3

Step to the next line in this file

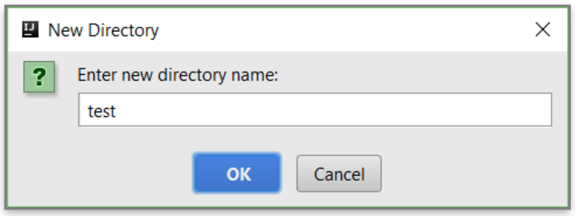
Run 'Config1'	Shift+F10
Debug 'Config1'	Shift+F9
Run 'Config1' with Coverage	
Run...	Alt+Shift+F10
Debug...	Alt+Shift+F9
Attach to Local Process...	
Edit Configurations...	
Import Test Results	
Stop 'Config1'	Ctrl+F2
Show Running List	
Reload Changed Classes	
Restart Activity	
Step Over	F8
Force Step Over	Alt+Shift+F8
Step Into	F7
Force Step Into	Alt+Shift+F7
Smart Step Into	Shift+F7
Step Out	Shift+F8
Run to Cursor	Alt+F9
Force Run to Cursor	Ctrl+Alt+F9
Drop Frame	
Pause Program	
Resume Program	F9
Evaluate Expression...	Alt+F8
Quick Evaluate Expression	Ctrl+Alt+F8
Show Execution Point	Alt+F10
Toggle Line Breakpoint	Ctrl+F8
Toggle Method Breakpoint	
Toggle Temporary Line Breakpoint	Ctrl+Alt+Shift+F8
Toggle Breakpoint Enabled	
View Breakpoints...	Ctrl+Shift+F8
Get thread dump	

Adding Libraries

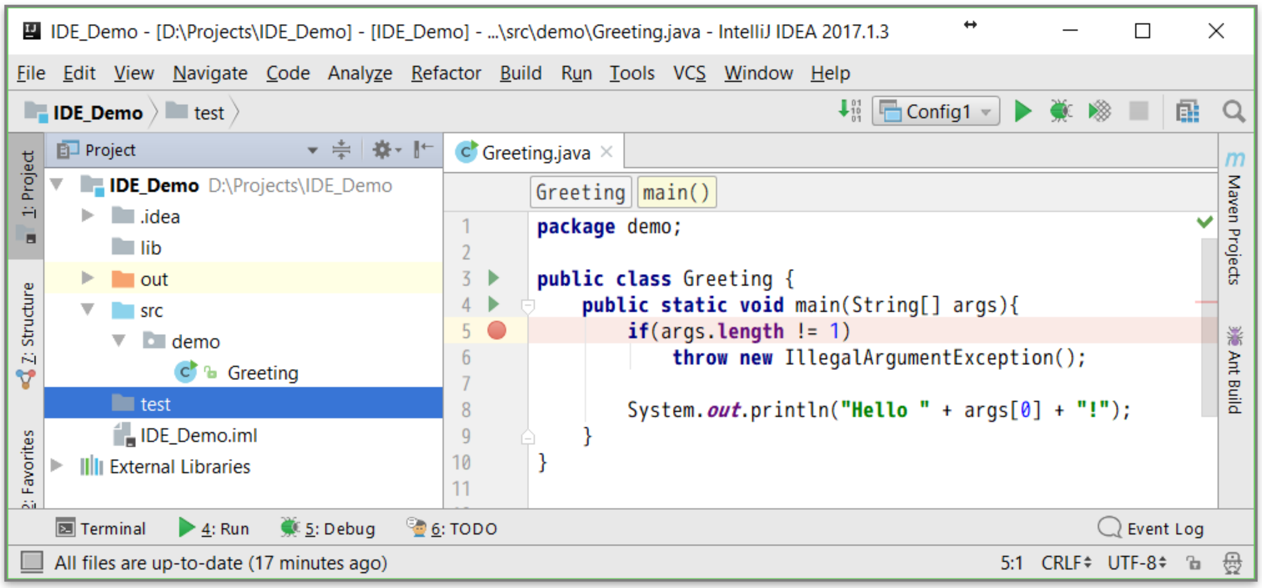
To add a new directory to the project, right click on the project node and select **New -> Directory**.


IDE_Demo - [D:\Projects\IDE_Demo] - [IDE_Demo] - ...\src\demo\Greeting.java - IntelliJ IDEA 2017.1.3
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
IDE_Demo src demo Greeting
Config1
Project
IDE_Demo D:\Projects\IDE_Demo
.idea
out
src
demo
Greeting
IDE_Demo.iml
External Libraries
Greeting.java
New
Add Framework Support...
Cut Ctrl+X
Copy Ctrl+C
Copy Path Ctrl+Shift+C
Copy as Plain Text
Copy Relative Path Ctrl+Alt+Shift+C
Paste Ctrl+V
Find Usages Alt+F7
Find in Path... Ctrl+Shift+F
Replace in Path... Ctrl+Shift+R
Module
File
Scratch File Ctrl+Alt+Shift+Insert
Directory
FXML File
HTML File
XSLT Stylesheet
Edit File Templates...
Resource Bundle
Terminal
4: Run
Event Log
Create new directory or pa
5:1 CRLF UTF-8
1: Project
7: Structure
2: Favorites
Maven Projects
Ant Build

Name the project as you wish and then click on **OK**.



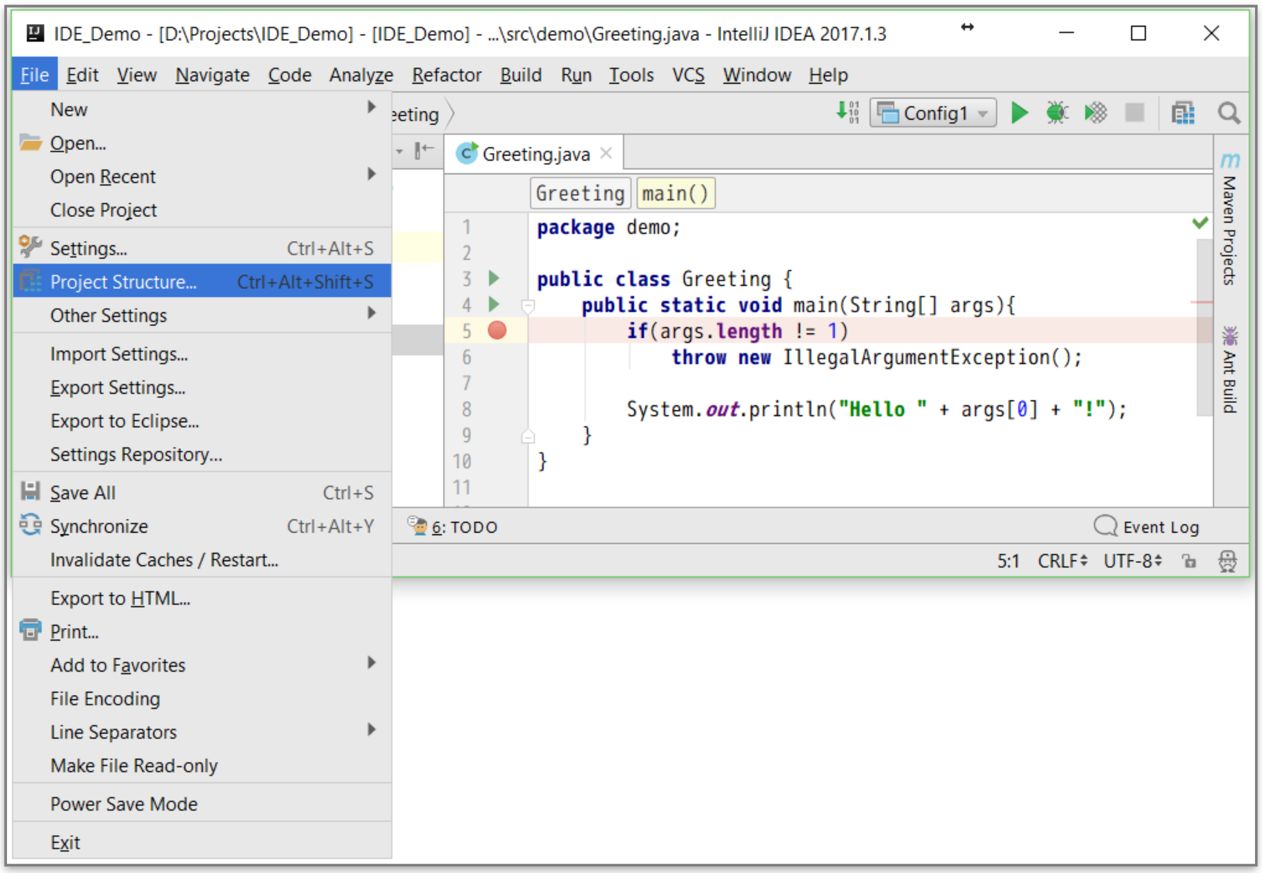
Add two directories **test** and **lib** to the project.



```
package demo;

public class Greeting {
    public static void main(String[] args){
        if(args.length != 1)
            throw new IllegalArgumentException();

        System.out.println("Hello " + args[0] + "!");
    }
}
```

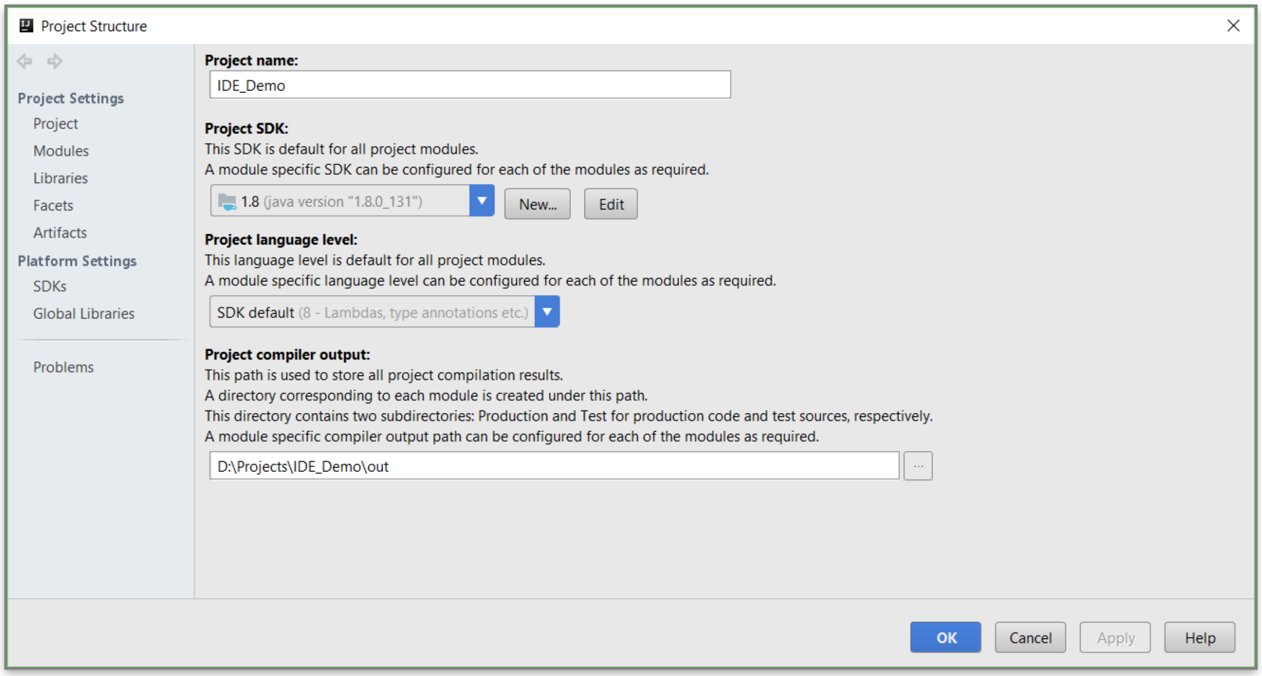
All files are up-to-date (17 minutes ago)   5:1 CRLF UTF-8

You can view the project structure by clicking on **File -> Project Structure**.



```
package demo;

public class Greeting {
    public static void main(String[] args){
        if(args.length != 1)
            throw new IllegalArgumentException();

        System.out.println("Hello " + args[0] + "!");
    }
}
```

In the **Project Structure** dialog, you can view/change some basic information about the project. Note Java compiler will put the compilation results into folder **out**.


Project Structure

Project Settings: Project, Modules, Libraries, Facets, Artifacts
Platform Settings: SDKs, Global Libraries
Problems

Project name:
IDE_Demo

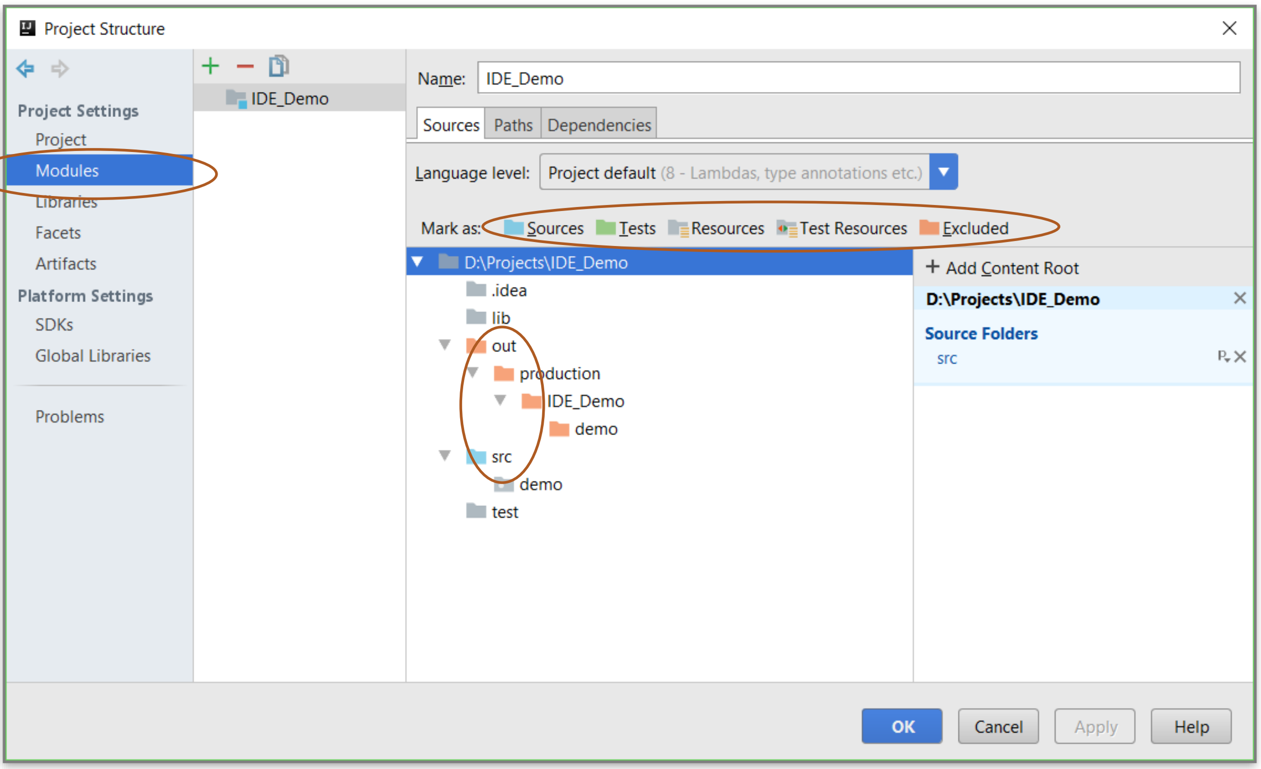
Project SDK:
This SDK is default for all project modules.
A module specific SDK can be configured for each of the modules as required.
1.8 (java version "1.8.0_131")   New...   Edit

Project language level:
This language level is default for all project modules.
A module specific language level can be configured for each of the modules as required.
SDK default (8 - Lambdas, type annotations etc.)

Project compiler output:
This path is used to store all project compilation results.
A directory corresponding to each module is created under this path.
This directory contains two subdirectories: Production and Test for production code and test sources, respectively.
A module specific compiler output path can be configured for each of the modules as required.
D:\Projects\IDE_Demo\out

OK   Cancel   Apply   Help

Select **Modules**, you should be able to see the directory structure of the project as the following. Note folder **src** is marked as **Sources** and folder **out** as **Excluded**.


Project Structure

Project Settings: Project, Modules, Libraries, Facets, Artifacts
Platform Settings: SDKs, Global Libraries
Problems

IDE_Demo

Name: IDE_Demo

Sources | Paths | Dependencies

Language level: Project default (8 - Lambdas, type annotations etc.)

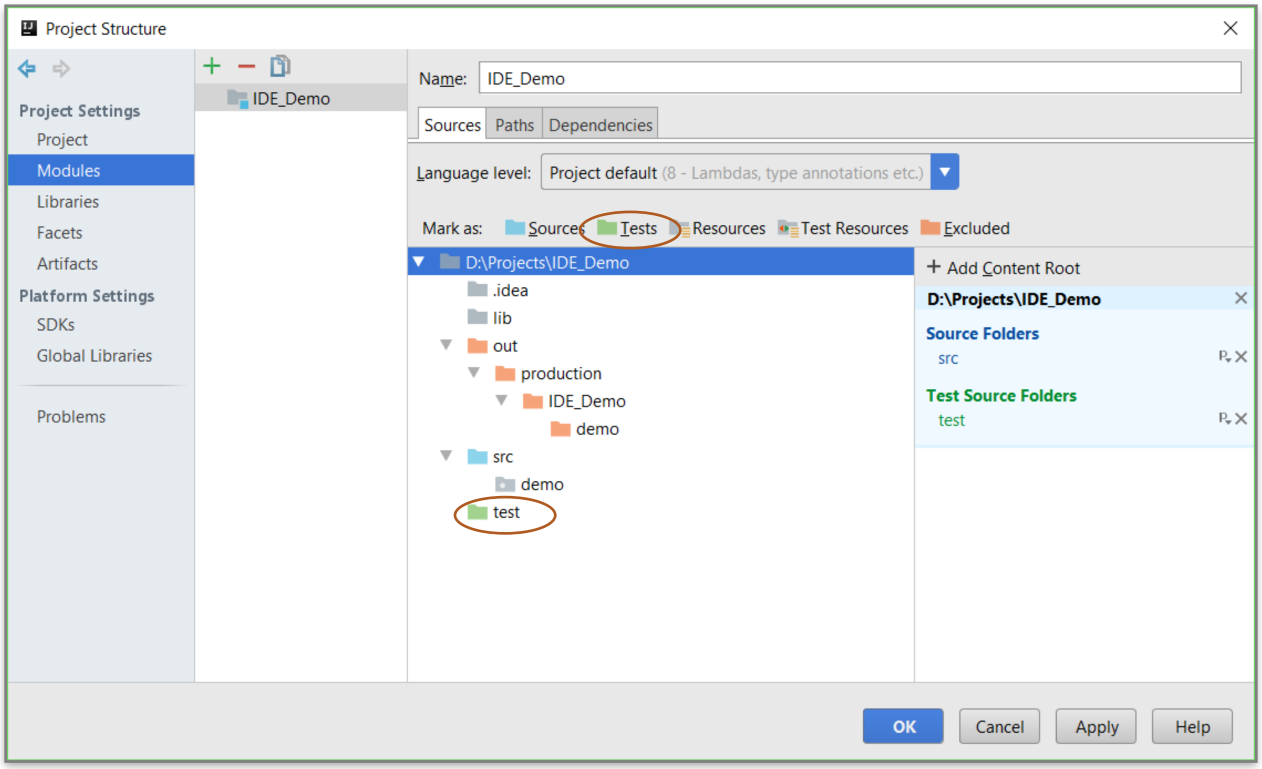
Mark as: Sources Tests Resources Test Resources Excluded

```
D:\Projects\IDE_Demo
    .idea
    lib
    out
        production
            IDE_Demo
                demo
    src
        demo
    test
```

+ Add Content Root
D:\Projects\IDE_Demo
Source Folders
src

OK   Cancel   Apply   Help

Select folder **test** and mark it as **Tests**. Afterwards, the structure should look like the following. Click on **OK** to save the changes.

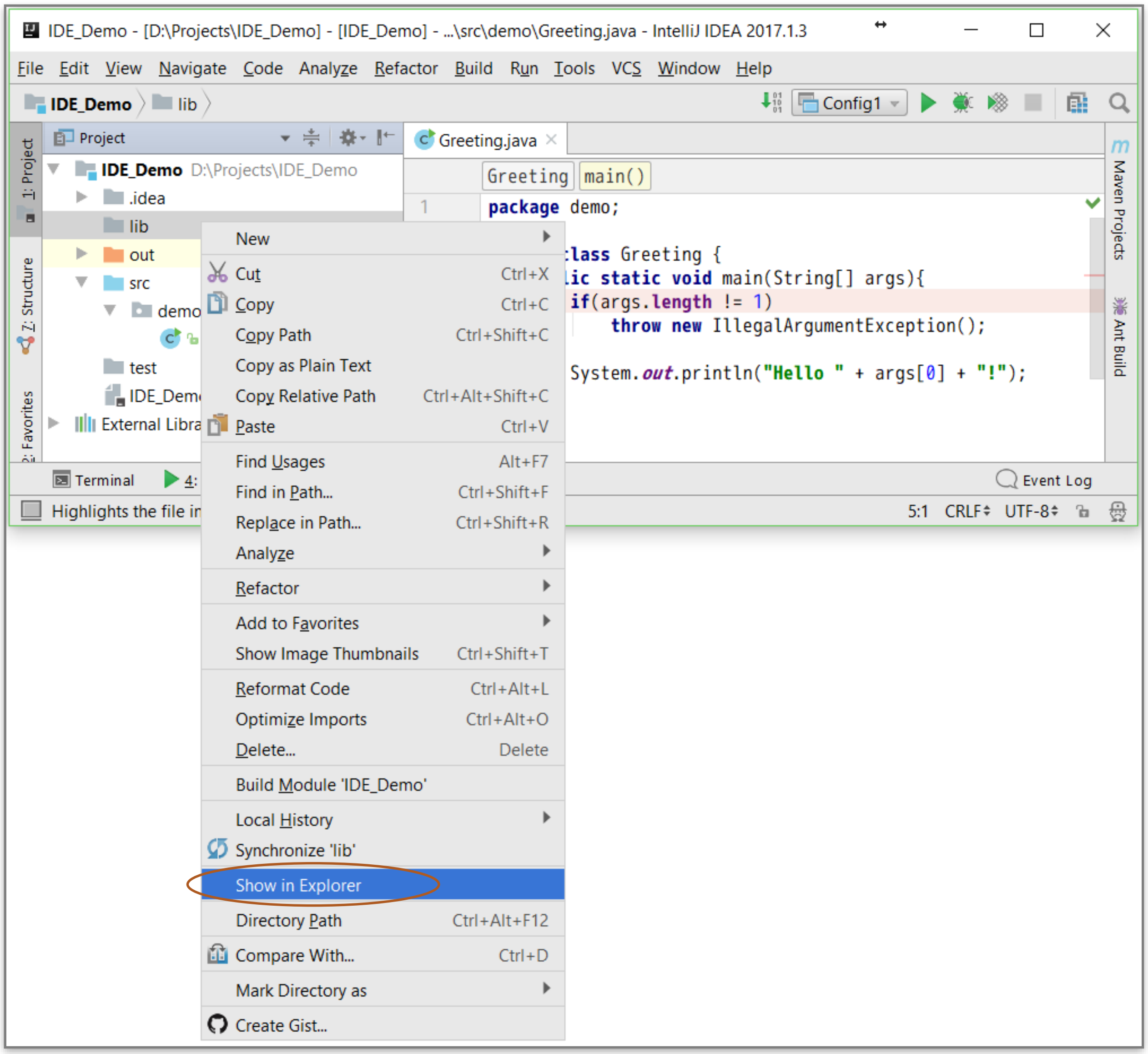


Right click on folder **lib** and then **Show in Explorer** to open the folder.

Copy the following two files from your IntelliJ IDEA installation to the **lib** folder. Note the exact path and file name may be slightly different depending on your system.

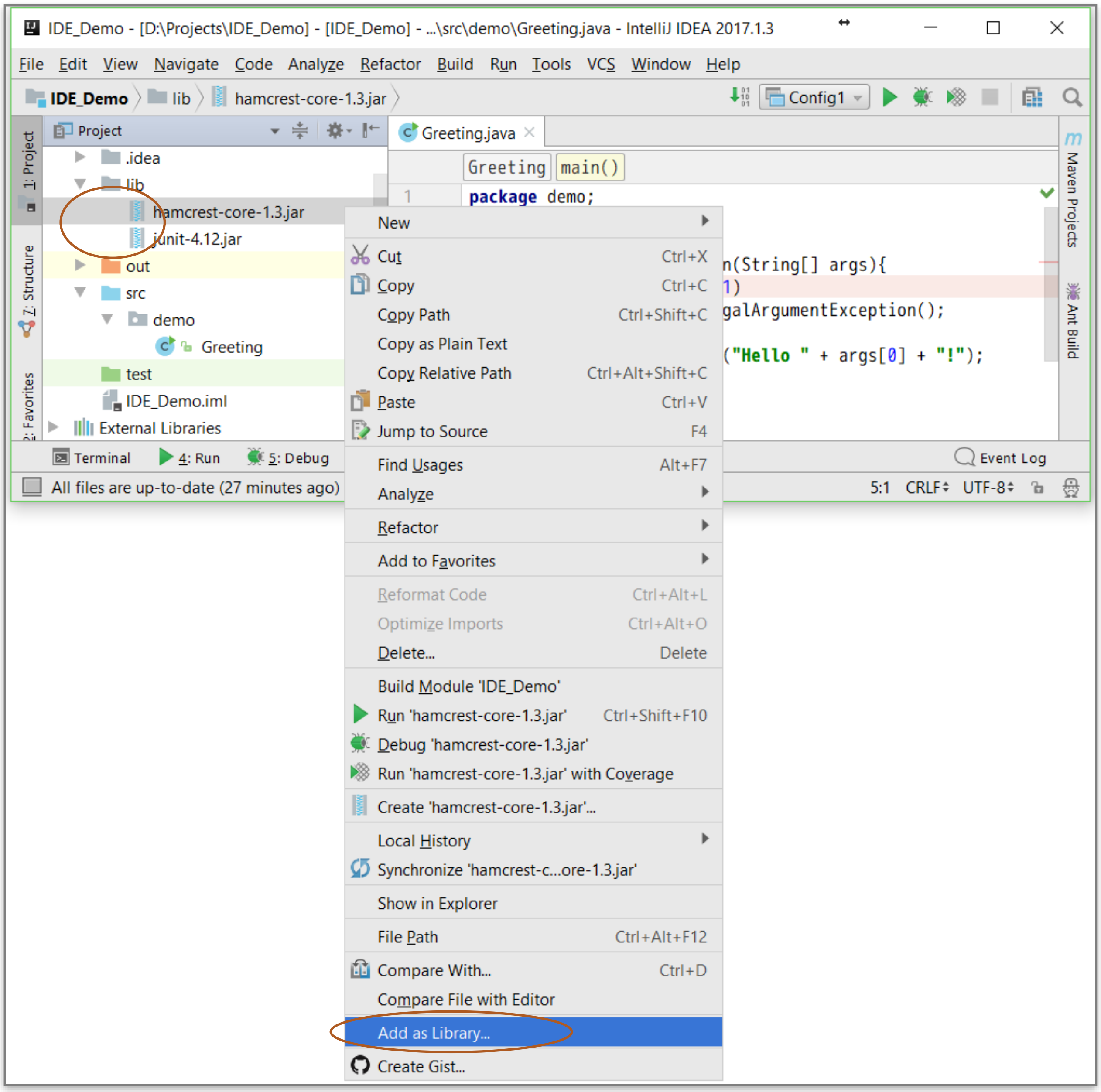
C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2017.1.3\lib\hamcrest-core-1.3.jar

C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2017.1.3\lib\junit-4.12.jar

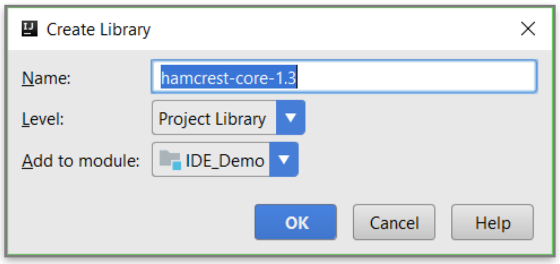


```
package demo;

:lass Greeting {
ic static void main(String[] args){
 if(args.length != 1)
     throw new IllegalArgumentException();

System.out.println("Hello " + args[0] + "!");
```

The two jar files should appear in folder **lib** in your IDE now, but they are not part of the project yet. Right click on a jar file and select **Add as Library…** and then click on **OK** in the pop up dialog. Do the same with the other jar file.

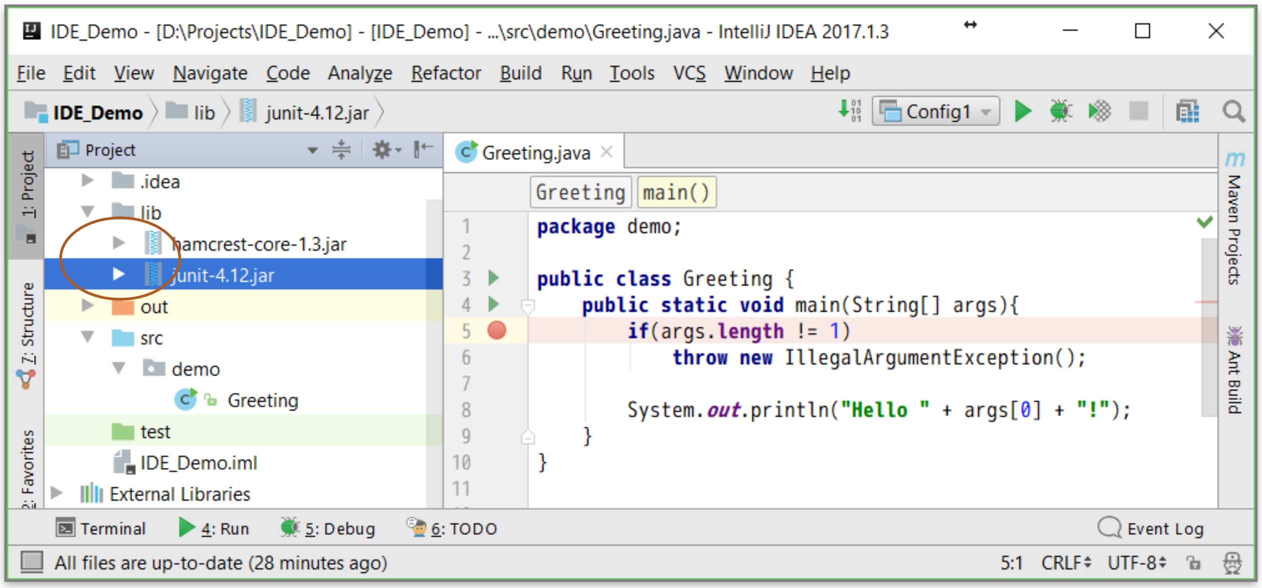


```
package demo;
n(String[] args){
1)
galArgumentException();
("Hello " + args[0] + "!");
```



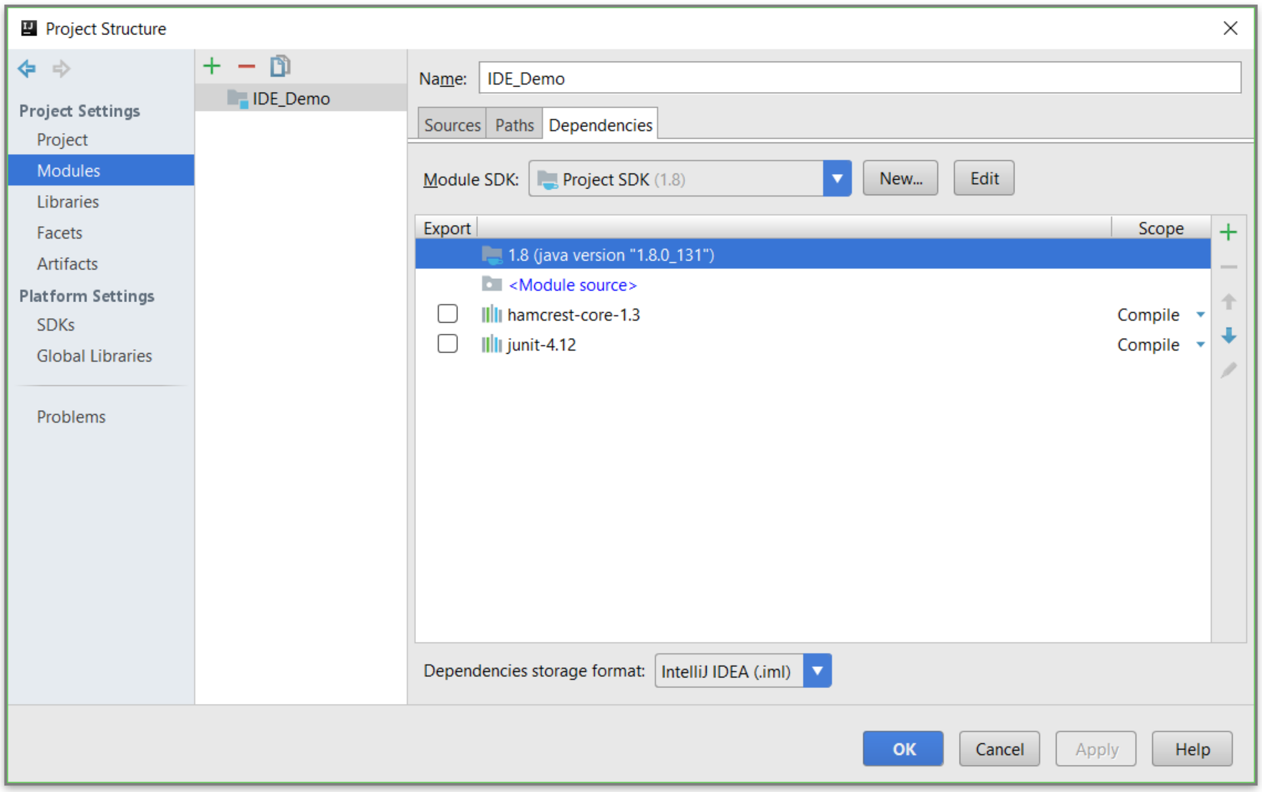
Name: hamcrest-core-1.3
Level: Project Library
Add to module: IDE_Demo

Now the two Jar files are added to the project as libraries. These two Jar files provides the JUnit testing framework.


IDE_Demo - [D:\Projects\IDE_Demo] - [IDE_Demo] - ...\src\demo\Greeting.java - IntelliJ IDEA 2017.1.3
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
IDE_Demo lib junit-4.12.jar
Config1
Project
.idea
lib
hamcrest-core-1.3.jar
junit-4.12.jar
out
src
demo
Greeting
test
IDE_Demo.iml
External Libraries
Greeting.java
Greeting main()
Terminal
4: Run
5: Debug
6: TODO
Event Log
All files are up-to-date (28 minutes ago)
5:1 CRLF UTF-8

```
package demo;

public class Greeting {
    public static void main(String[] args){
        if(args.length != 1)
            throw new IllegalArgumentException();

        System.out.println("Hello " + args[0] + "!");
    }
}
```

If you open **Project Structure** again, you will see the two Jar files have indeed been added as libraries.



Name: IDE_Demo

Module SDK: Project SDK (1.8)

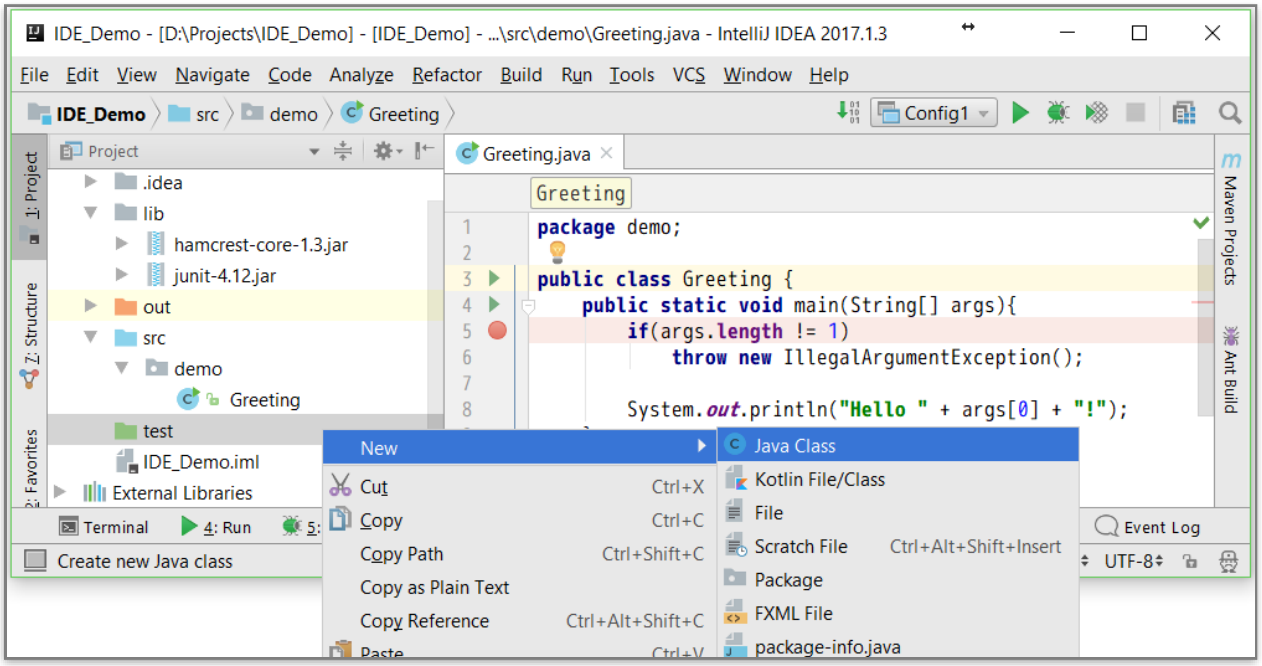
Export		Scope
	1.8 (java version "1.8.0_131")	
	<Module source>	
☐	hamcrest-core-1.3	Compile
☐	junit-4.12	Compile

Dependencies storage format: IntelliJ IDEA (.iml)

Testing Your Code

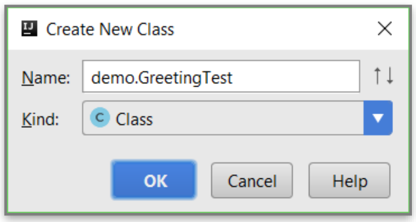
Detailed instructions at IntelliJ IDEA Help: https://www.jetbrains.com/help/idea/create-tests.html

Next, we create a JUnit test class by right clicking on the test folder and then selecting **New -> Java Class**.



```java
package demo;

public class Greeting {
    public static void main(String[] args){
        if(args.length != 1)
            throw new IllegalArgumentException();

        System.out.println("Hello " + args[0] + "!");
```

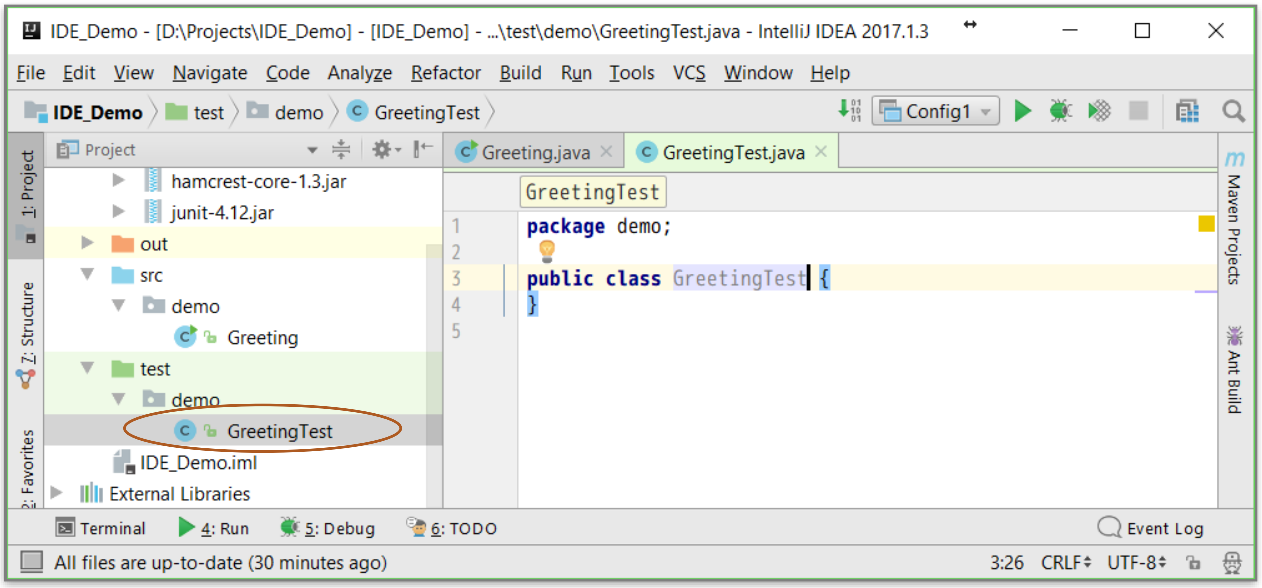
Provide the fully qualified name for your test class, then click on **OK**.



Name: demo.GreetingTest

Kind: Class

Double click on the test class name to open its source in Editor.

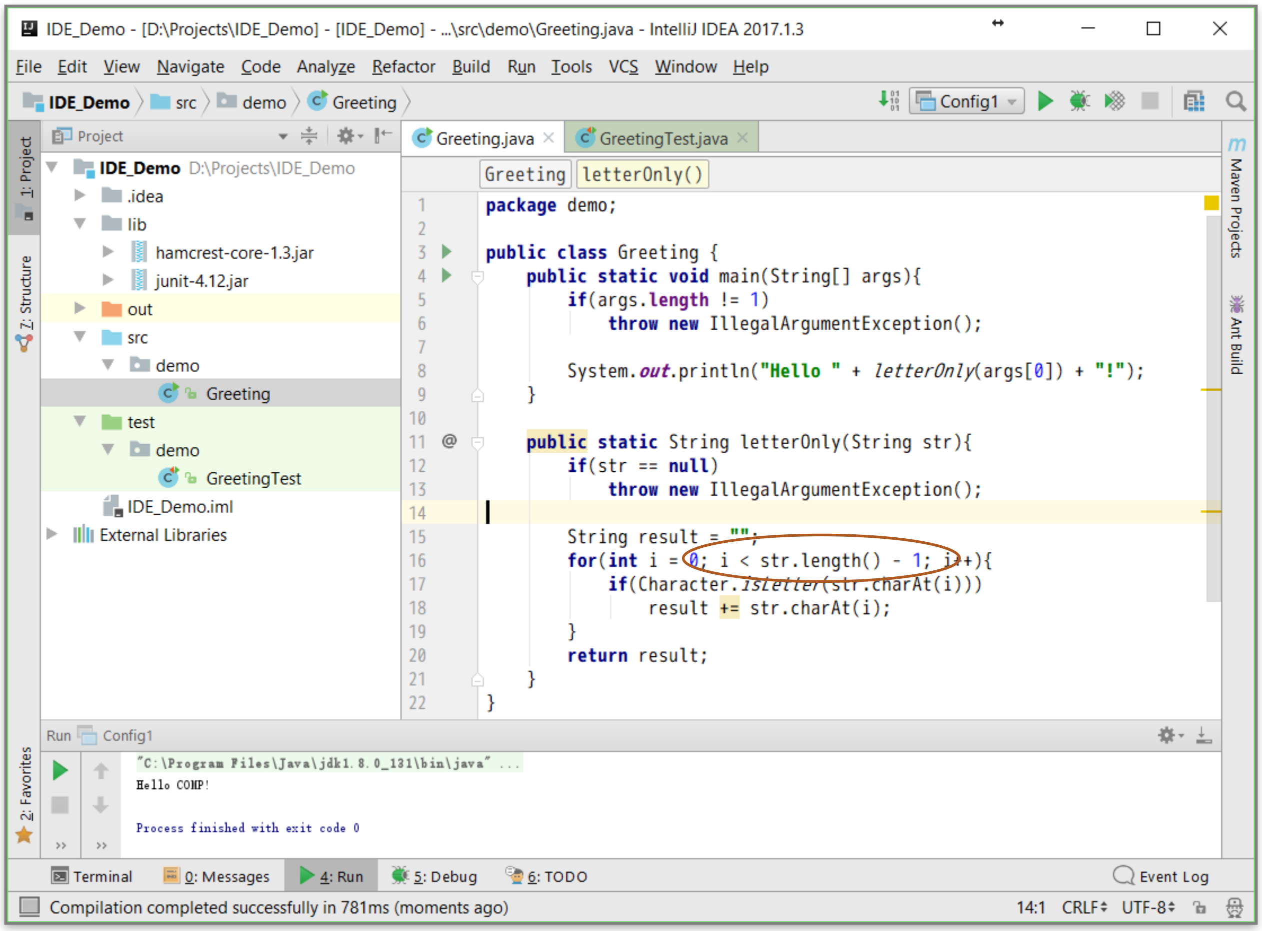


```
package demo;

public class GreetingTest {
}
```

All files are up-to-date (30 minutes ago)   3:26 CRLF UTF-8

Before writing any test method, we first add a new **public static** method to class **Greeting**. The method takes a **String** and returns a new **String** with only letters from the argument left. The **main** method of the application invokes the new method to process its argument before printing it out.

Note the loop-continuation condition in the implementation is wrong, and the correct condition should be **i<str.length()**. The mistake will only cause a problem if the argument ends with a letter.

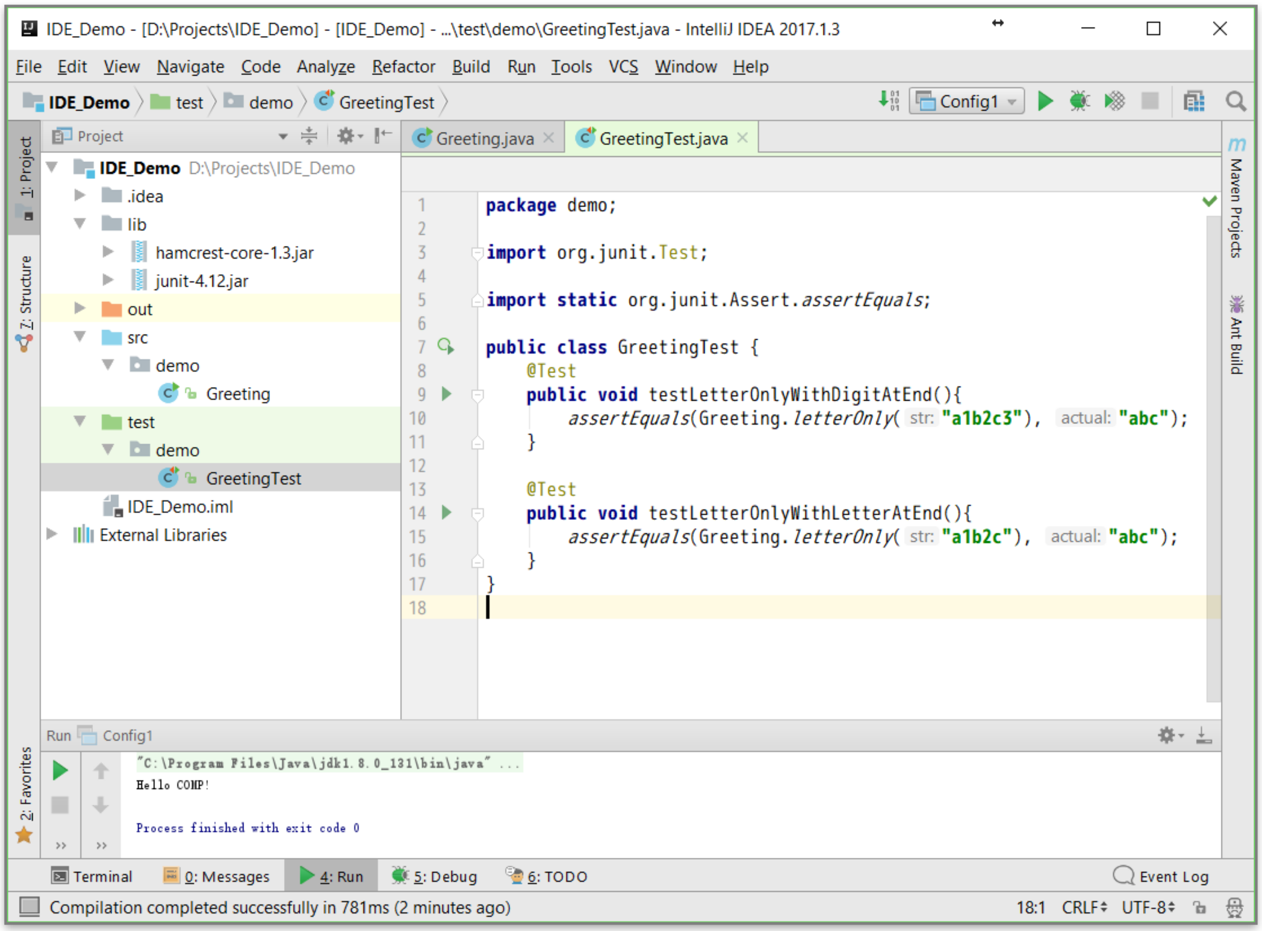

IDE_Demo - [D:\Projects\IDE_Demo] - [IDE_Demo] - ...\src\demo\Greeting.java - IntelliJ IDEA 2017.1.3
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
IDE_Demo src demo Greeting Config1
Project
IDE_Demo D:\Projects\IDE_Demo
.idea
lib
hamcrest-core-1.3.jar
junit-4.12.jar
out
src
demo
Greeting
test
demo
GreetingTest
IDE_Demo.iml
External Libraries
Greeting.java GreetingTest.java
Greeting letterOnly()
Maven Projects
Ant Build
1: Project
Z: Structure
2: Favorites
Run Config1
Terminal 0: Messages 4: Run 5: Debug 6: TODO Event Log
14:1 CRLF UTF-8

```
package demo;

public class Greeting {
    public static void main(String[] args){
        if(args.length != 1)
            throw new IllegalArgumentException();

        System.out.println("Hello " + letterOnly(args[0]) + "!");
    }

    public static String letterOnly(String str){
        if(str == null)
            throw new IllegalArgumentException();

        String result = "";
        for(int i = 0; i < str.length() - 1; i++){
            if(Character.isLetter(str.charAt(i)))
                result += str.charAt(i);
        }
        return result;
    }
}
```

```
"C:\Program Files\Java\jdk1.8.0_131\bin\java" ...
Hello COMP!

Process finished with exit code 0
```

Compilation completed successfully in 781ms (moments ago)

Next, we write two tests for the new method, one using a string that ends with a digit and the other a string that ends with a letter. Annotation **@Test** turns a **public void** argument-less method into a test method, while **assertEquals** will use **equals()** method to compare its two arguments and throw an exception if they are not equal.

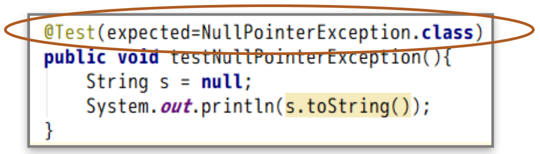


```
package demo;

import org.junit.Test;

import static org.junit.Assert.assertEquals;

public class GreetingTest {
    @Test
    public void testLetterOnlyWithDigitAtEnd(){
        assertEquals(Greeting.letterOnly( str: "a1b2c3"),  actual: "abc");
    }

    @Test
    public void testLetterOnlyWithLetterAtEnd(){
        assertEquals(Greeting.letterOnly( str: "a1b2c"),  actual: "abc");
    }
}
```

```
"C:\Program Files\Java\jdk1.8.0_131\bin\java" ...
Hello COMP!

Process finished with exit code 0
```

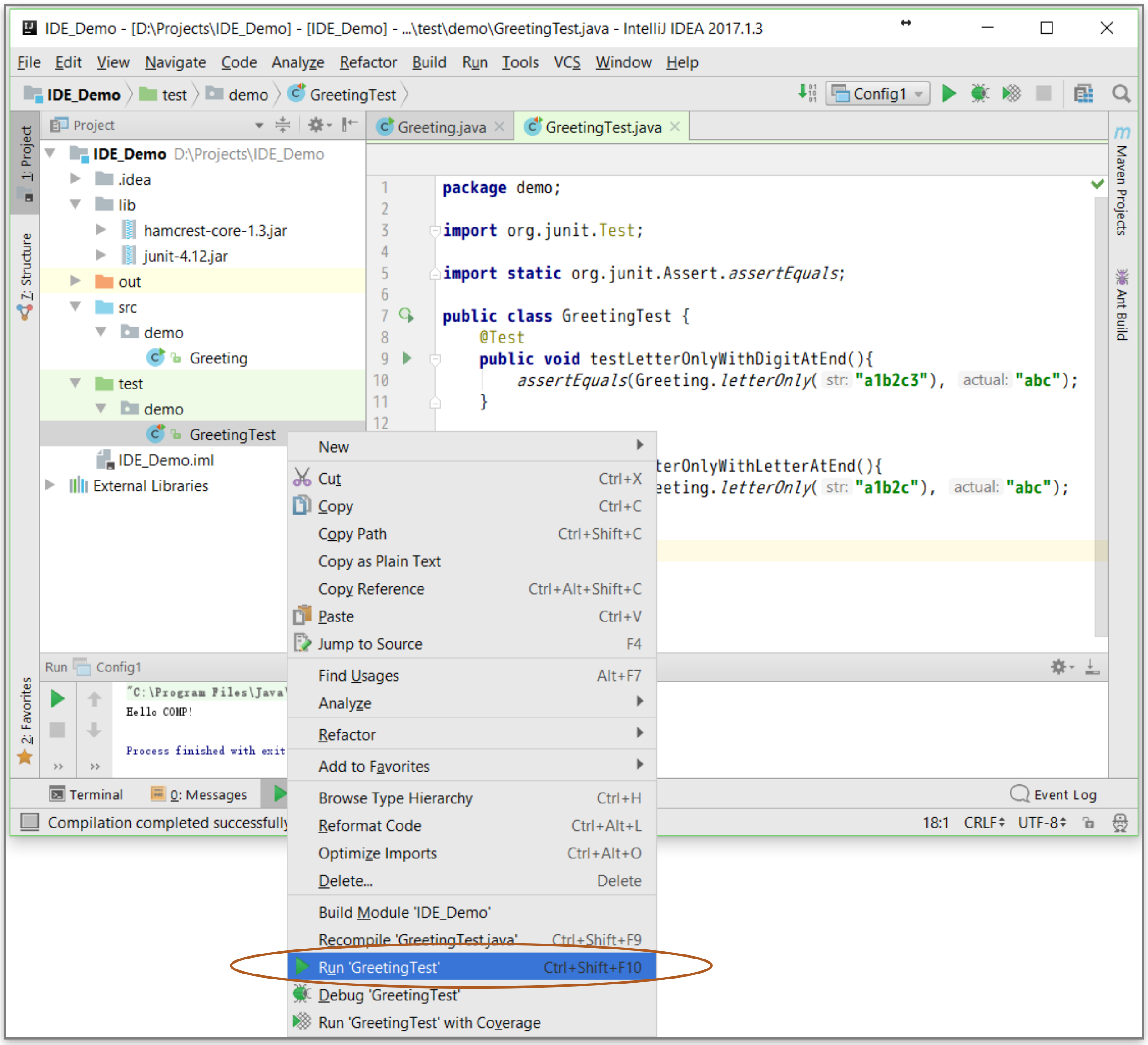
Compilation completed successfully in 781ms (2 minutes ago)

Tests that terminate without exceptions are in general considered successful, but JUnit does provide a way to specify that a test is expected to terminate with an exception of specific type, as shown in the example below.



```
@Test(expected=NullPointerException.class)
public void testNullPointerException(){
    String s = null;
    System.out.println(s.toString());
}
```

Once we have finished the test class, we can right click on the test class and select **Run 'GreetingTest'** to run the tests. If you prefer, you can also right click on a test method and run only that single test.

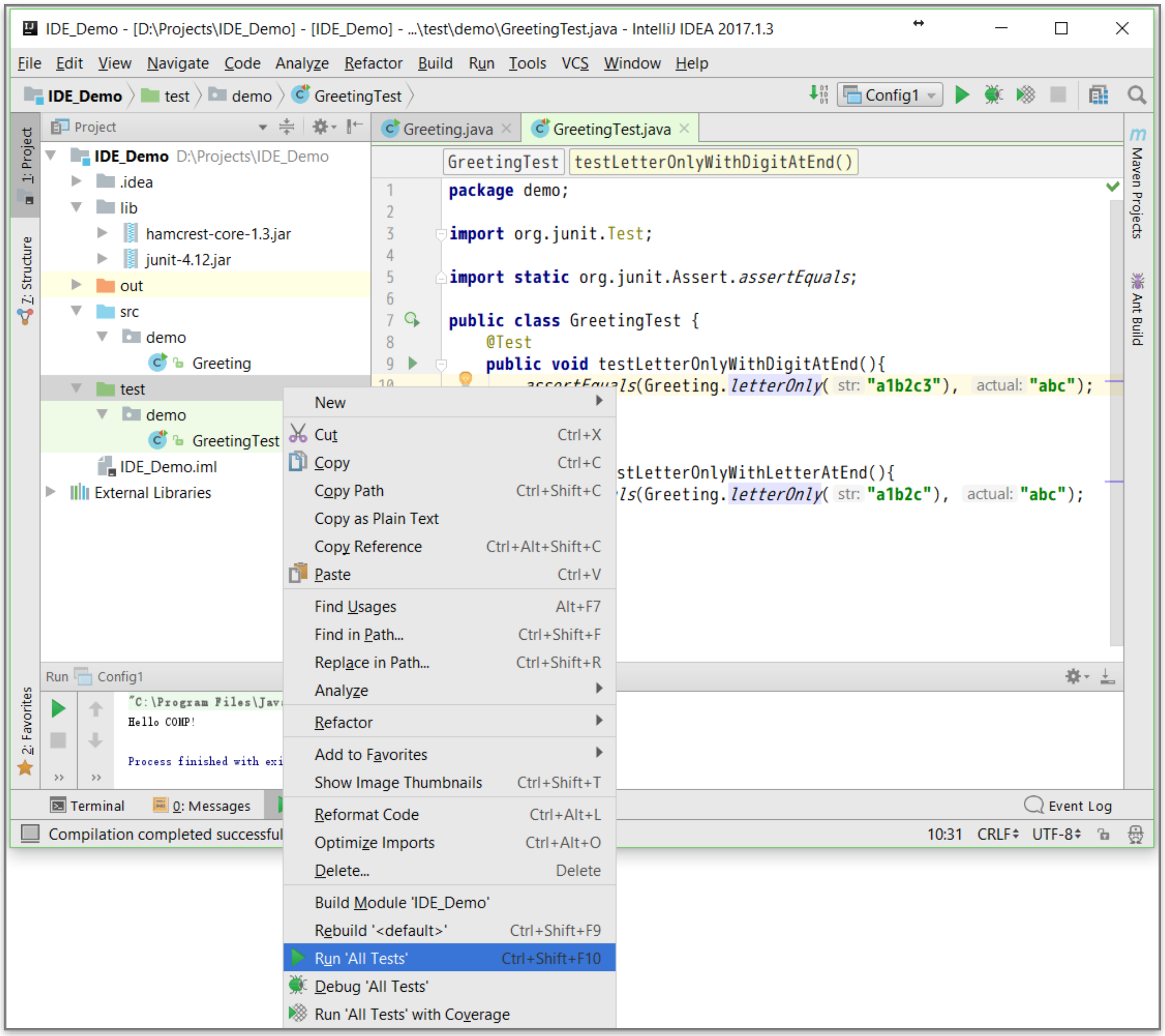


```java
package demo;

import org.junit.Test;

import static org.junit.Assert.assertEquals;

public class GreetingTest {
    @Test
    public void testLetterOnlyWithDigitAtEnd(){
        assertEquals(Greeting.letterOnly( str: "a1b2c3"),  actual: "abc");
    }

...terOnlyWithLetterAtEnd(){
...eeting.letterOnly( str: "a1b2c"),  actual: "abc");
```

```
"C:\Program Files\Java\
Hello COMP!

Process finished with exit
```

IDE_Demo - [D:\Projects\IDE_Demo] - [IDE_Demo] - ...\test\demo\GreetingTest.java - IntelliJ IDEA 2017.1.3
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
IDE_Demo test demo GreetingTest Config1
Project
IDE_Demo D:\Projects\IDE_Demo
.idea
lib
hamcrest-core-1.3.jar
junit-4.12.jar
out
src
demo
Greeting
test
demo
GreetingTest
IDE_Demo.iml
External Libraries
Greeting.java GreetingTest.java
New
Cut Ctrl+X
Copy Ctrl+C
Copy Path Ctrl+Shift+C
Copy as Plain Text
Copy Reference Ctrl+Alt+Shift+C
Paste Ctrl+V
Jump to Source F4
Find Usages Alt+F7
Analyze
Refactor
Add to Favorites
Browse Type Hierarchy Ctrl+H
Reformat Code Ctrl+Alt+L
Optimize Imports Ctrl+Alt+O
Delete... Delete
Build Module 'IDE_Demo'
Recompile 'GreetingTest.java' Ctrl+Shift+F9
Run 'GreetingTest' Ctrl+Shift+F10
Debug 'GreetingTest'
Run 'GreetingTest' with Coverage
Run Config1
Terminal 0: Messages
Event Log
Compilation completed successfully
18:1 CRLF UTF-8
1: Project
7: Structure
2: Favorites
Maven Projects
Ant Build

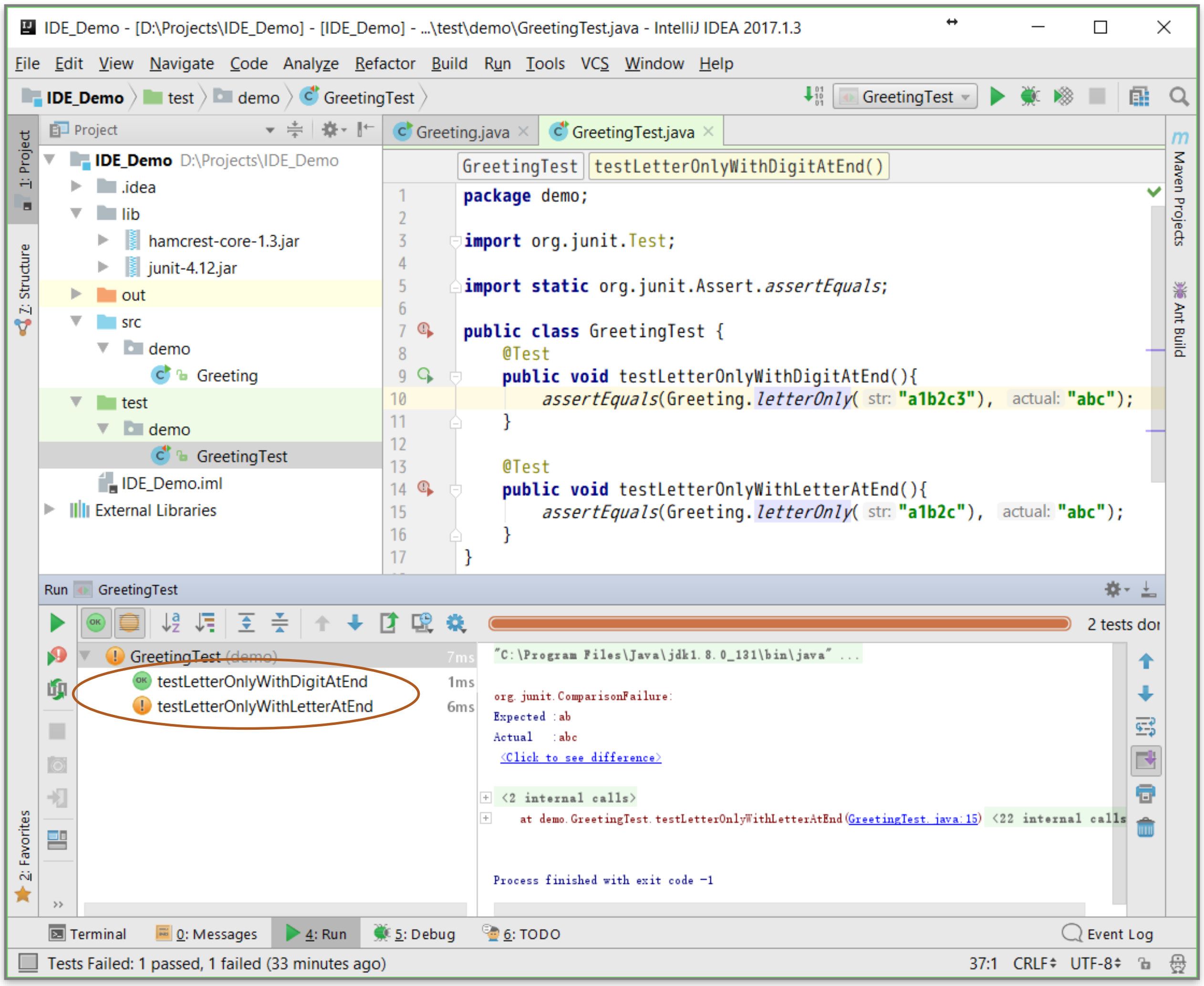
In case you have multiple test classes and want to run all of them, you can right click on folder test and select **Run 'All Tests'**.


IDE_Demo - [D:\Projects\IDE_Demo] - [IDE_Demo] - ...\test\demo\GreetingTest.java - IntelliJ IDEA 2017.1.3
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
IDE_Demo test demo GreetingTest
Config1
Project
IDE_Demo D:\Projects\IDE_Demo
.idea
lib
hamcrest-core-1.3.jar
junit-4.12.jar
out
src
demo
Greeting
test
demo
GreetingTest
IDE_Demo.iml
External Libraries
Greeting.java
GreetingTest.java
GreetingTest
testLetterOnlyWithDigitAtEnd()
New
Cut Ctrl+X
Copy Ctrl+C
Copy Path Ctrl+Shift+C
Copy as Plain Text
Copy Reference Ctrl+Alt+Shift+C
Paste Ctrl+V
Find Usages Alt+F7
Find in Path... Ctrl+Shift+F
Replace in Path... Ctrl+Shift+R
Analyze
Refactor
Add to Favorites
Show Image Thumbnails Ctrl+Shift+T
Reformat Code Ctrl+Alt+L
Optimize Imports Ctrl+Alt+O
Delete... Delete
Build Module 'IDE_Demo'
Rebuild '<default>' Ctrl+Shift+F9
Run 'All Tests' Ctrl+Shift+F10
Debug 'All Tests'
Run 'All Tests' with Coverage
Run Config1
Terminal
0: Messages
Event Log
Compilation completed successful
10:31 CRLF UTF-8
1: Project
7: Structure
2: Favorites
Maven Projects
Ant Build

```java
package demo;

import org.junit.Test;

import static org.junit.Assert.assertEquals;

public class GreetingTest {
    @Test
    public void testLetterOnlyWithDigitAtEnd(){
        assertEquals(Greeting.letterOnly( str: "a1b2c3"),  actual: "abc");

stLetterOnlyWithLetterAtEnd(){
ls(Greeting.letterOnly( str: "a1b2c"),  actual: "abc");
```

```
"C:\Program Files\Jav
Hello COMP!

Process finished with exi
```

JUnit will run the tests automatically and report the results. You can easily see which tests have passed and which ones have failed. Note that each test method is a test.


IDE_Demo - [D:\Projects\IDE_Demo] - [IDE_Demo] - ...\test\demo\GreetingTest.java - IntelliJ IDEA 2017.1.3
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
IDE_Demo test demo GreetingTest
GreetingTest
Project
IDE_Demo D:\Projects\IDE_Demo
.idea
lib
hamcrest-core-1.3.jar
junit-4.12.jar
out
src
demo
Greeting
test
demo
GreetingTest
IDE_Demo.iml
External Libraries
Greeting.java
GreetingTest.java
GreetingTest testLetterOnlyWithDigitAtEnd()
Run GreetingTest
2 tests dor
Terminal 0: Messages 4: Run 5: Debug 6: TODO Event Log
37:1 CRLF UTF-8

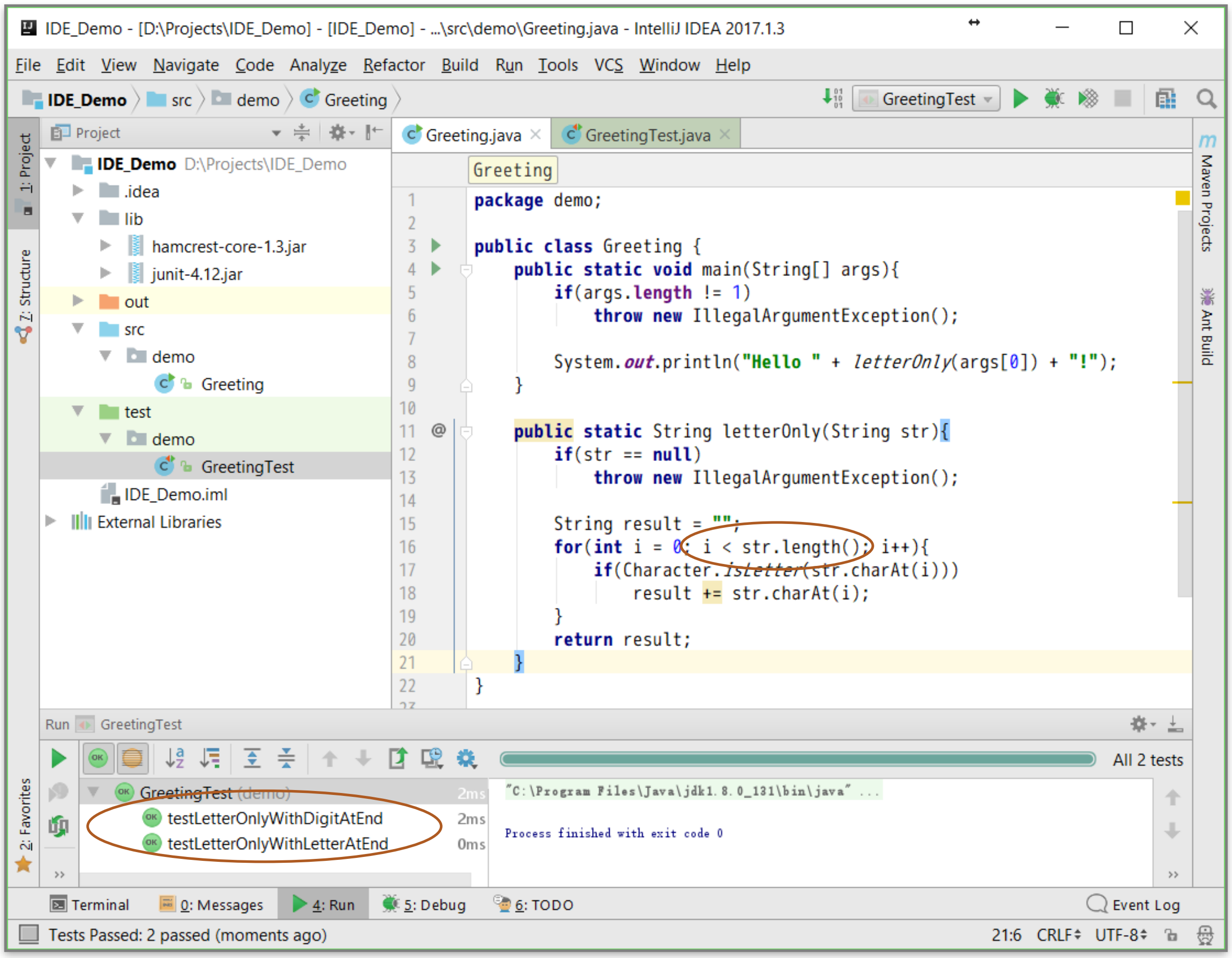
```java
package demo;

import org.junit.Test;

import static org.junit.Assert.assertEquals;

public class GreetingTest {
    @Test
    public void testLetterOnlyWithDigitAtEnd(){
        assertEquals(Greeting.letterOnly( str: "a1b2c3"),  actual: "abc");
    }

    @Test
    public void testLetterOnlyWithLetterAtEnd(){
        assertEquals(Greeting.letterOnly( str: "a1b2c"),  actual: "abc");
    }
}
```

```
GreetingTest (demo)                7ms
    testLetterOnlyWithDigitAtEnd   1ms
    testLetterOnlyWithLetterAtEnd  6ms
```

```
"C:\Program Files\Java\jdk1.8.0_131\bin\java" ...

org.junit.ComparisonFailure:
Expected :ab
Actual   :abc
 <Click to see difference>

+ <2 internal calls>
+   at demo.GreetingTest.testLetterOnlyWithLetterAtEnd(GreetingTest.java:15) <22 internal calls


Process finished with exit code -1
```

Tests Failed: 1 passed, 1 failed (33 minutes ago)

If we correct the mistake in method **letterOnly** and run the tests again, we will see all the test become passing.


IDE_Demo - [D:\Projects\IDE_Demo] - [IDE_Demo] - ...\src\demo\Greeting.java - IntelliJ IDEA 2017.1.3
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
IDE_Demo src demo Greeting
GreetingTest
Project
IDE_Demo D:\Projects\IDE_Demo
.idea
lib
hamcrest-core-1.3.jar
junit-4.12.jar
out
src
demo
Greeting
test
demo
GreetingTest
IDE_Demo.iml
External Libraries
Greeting.java
GreetingTest.java
Greeting

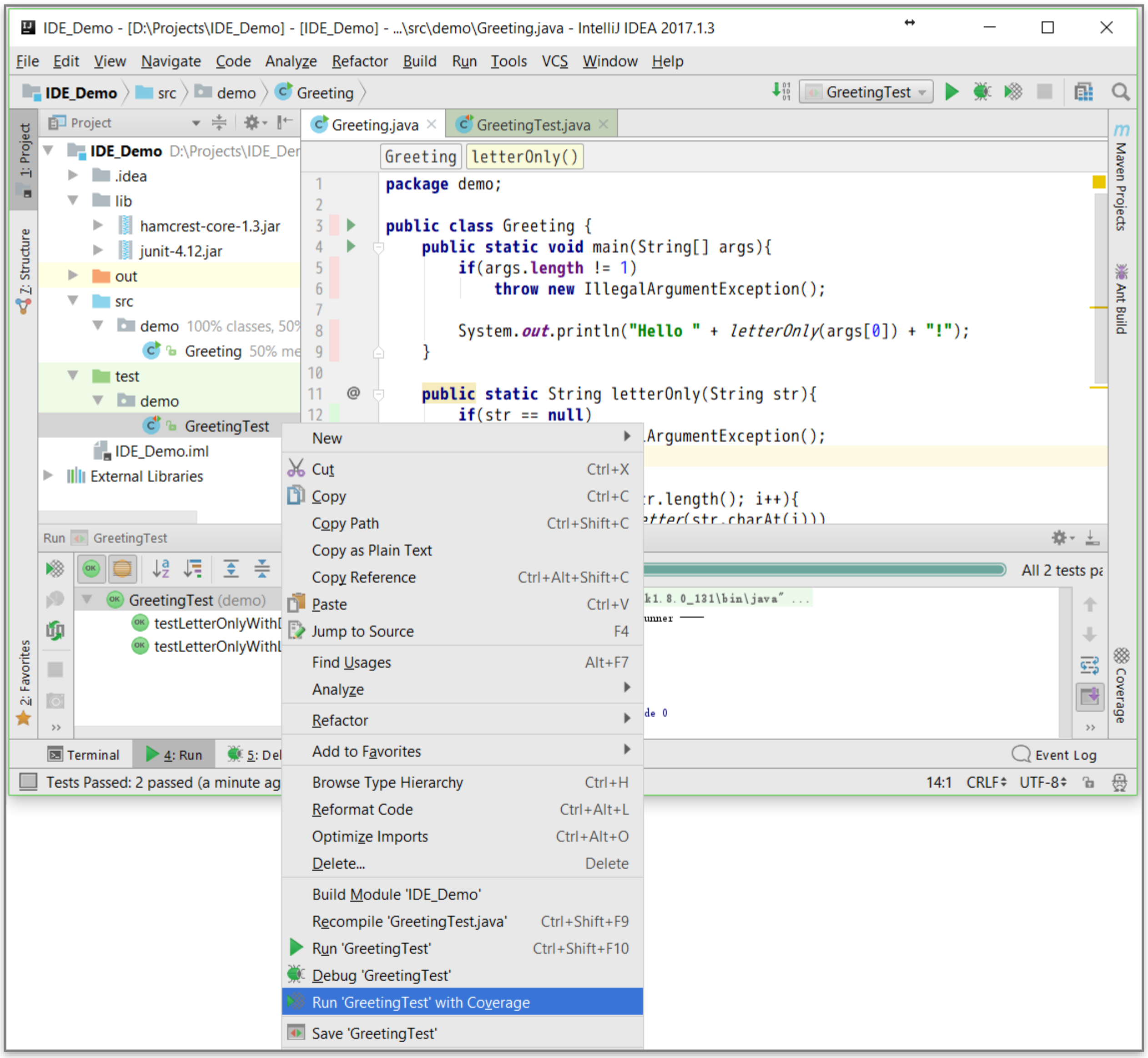
```
package demo;

public class Greeting {
    public static void main(String[] args){
        if(args.length != 1)
            throw new IllegalArgumentException();

        System.out.println("Hello " + letterOnly(args[0]) + "!");
    }

    public static String letterOnly(String str){
        if(str == null)
            throw new IllegalArgumentException();

        String result = "";
        for(int i = 0; i < str.length(); i++){
            if(Character.isLetter(str.charAt(i)))
                result += str.charAt(i);
        }
        return result;
    }
}
```

```
"C:\Program Files\Java\jdk1.8.0_131\bin\java" ...

Process finished with exit code 0
```

Tests Passed: 2 passed (moments ago)

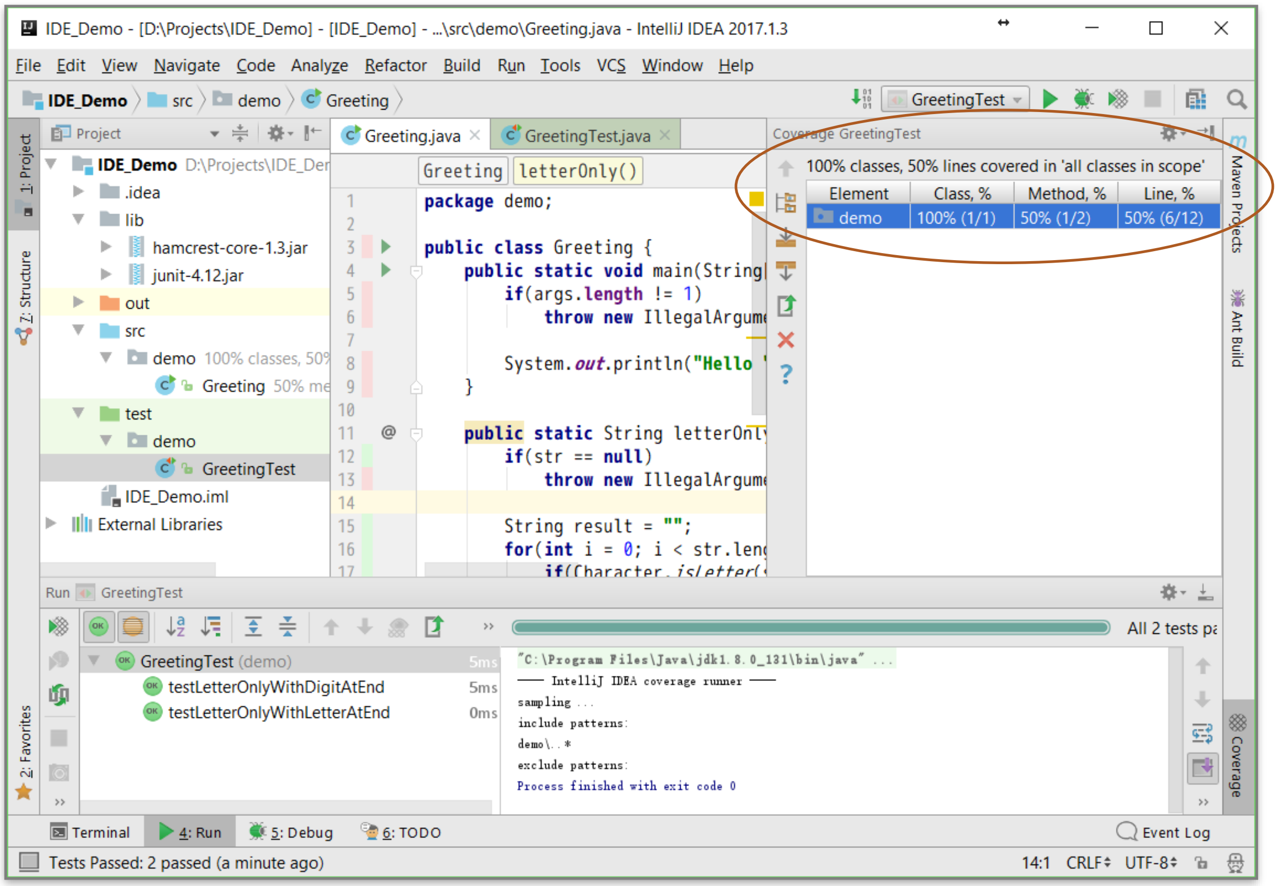
Code coverage of a group of tests measures how many percentage of code is exercised by those tests, and it gives programmer an idea about how thorough the group of tests is. Tests achieving 90% code coverage are more thorough than those achieving 30%, and, if both tests find no problems, we gain more confidence from the first than the second.

To measure the code coverage of the tests, we run the tests with Coverage.



```
package demo;

public class Greeting {
    public static void main(String[] args){
        if(args.length != 1)
            throw new IllegalArgumentException();

        System.out.println("Hello " + letterOnly(args[0]) + "!");
    }

    public static String letterOnly(String str){
        if(str == null)
                    .ArgumentException();

                    r.length(); i++){
                    etter(str.charAt(i)))
```

When testing is finished, coverage information will be shown in the **Coverage** tool window.



```
package demo;

public class Greeting {
    public static void main(String
        if(args.length != 1)
            throw new IllegalArgume

        System.out.println("Hello '
    }

    public static String letterOnl
        if(str == null)
            throw new IllegalArgume

        String result = "";
        for(int i = 0; i < str.len
            if(Character.isLetter(
```

100% classes, 50% lines covered in 'all classes in scope'

Element	Class, %	Method, %	Line, %
demo	100% (1/1)	50% (1/2)	50% (6/12)

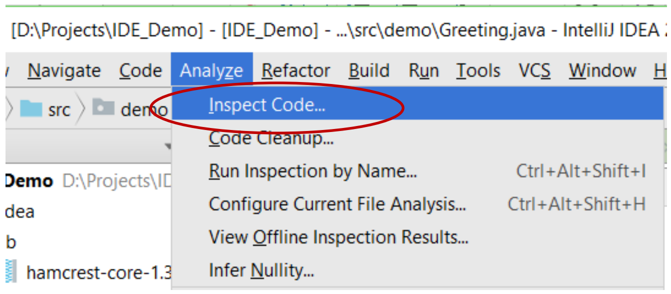
GreetingTest (demo)	5ms
testLetterOnlyWithDigitAtEnd	5ms
testLetterOnlyWithLetterAtEnd	0ms

```
"C:\Program Files\Java\jdk1.8.0_131\bin\java" ...
---- IntelliJ IDEA coverage runner ----
sampling ...
include patterns:
demo\..*
exclude patterns:
Process finished with exit code 0
```

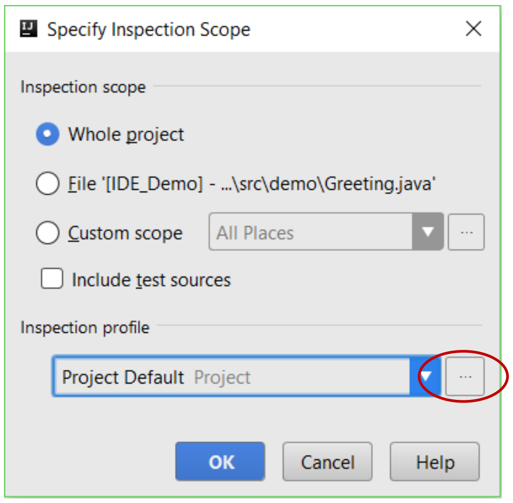
Tests Passed: 2 passed (a minute ago)   14:1 CRLF UTF-8

Running The Inspection Tool

Go to **Analyze** -> **Inspect Code…**


[D:\Projects\IDE_Demo] - [IDE_Demo] - …\src\demo\Greeting.java - IntelliJ IDEA
Navigate Code Analyze Refactor Build Run Tools VCS Window H
src demo
Demo D:\Projects\ID
dea
b
hamcrest-core-1.3
Inspect Code…
Code Cleanup…
Run Inspection by Name… Ctrl+Alt+Shift+I
Configure Current File Analysis… Ctrl+Alt+Shift+H
View Offline Inspection Results…
Infer Nullity…

In the dialog, select the **Inspection scope** and uncheck **Include test sources**. Then click on the ellipses at the bottom-right corner of the dialog.

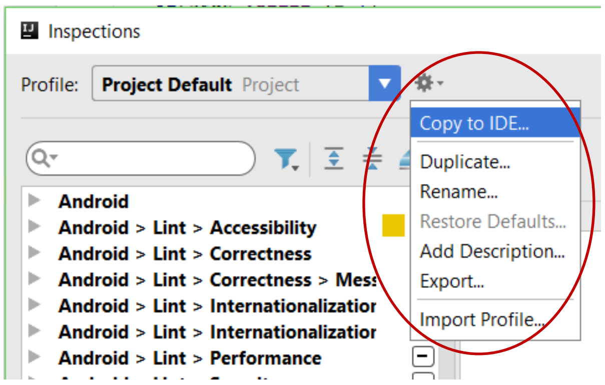


Inspection scope

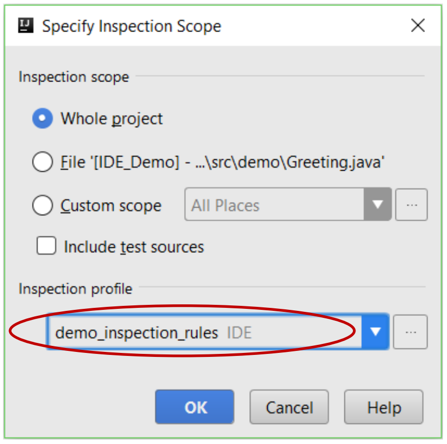
- [✓ Whole project]
- File '[IDE_Demo] - …\src\demo\Greeting.java'
- Custom scope: All Places
- Include test sources: unchecked

Inspection profile: Project Default Project

Click on the gear button, then **Import Profile…**


Inspections
Profile: Project Default Project
Copy to IDE…
Duplicate…
Rename…
Restore Defaults…
Add Description…
Export…
Import Profile…
Android
Android > Lint > Accessibility
Android > Lint > Correctness
Android > Lint > Correctness > Mess
Android > Lint > Internationalizatior
Android > Lint > Internationalizatior
Android > Lint > Performance

Import the desired inspection rules from an XML file and click on **OK** to go back to the previous dialog. Select the imported profile as the active **Inspection profile**.


Specify Inspection Scope

Inspection scope

(●) Whole project

( ) File '[IDE_Demo] - ...\src\demo\Greeting.java'

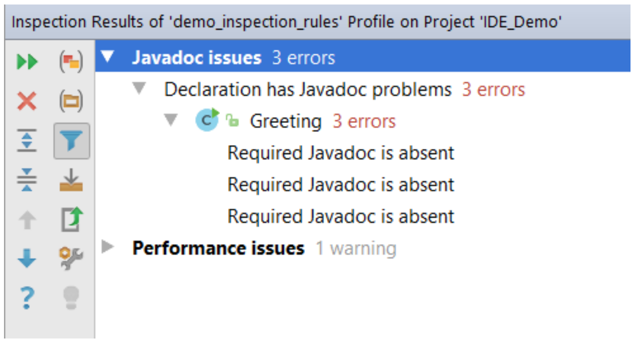
( ) Custom scope: All Places

[ ] Include test sources

Inspection profile

demo_inspection_rules IDE

Click on **OK** to run the inspection. Inspection results will be shown in the **Inspection Results** tool panel.


```
Inspection Results of 'demo_inspection_rules' Profile on Project 'IDE_Demo'
▼ Javadoc issues 3 errors
    ▼ Declaration has Javadoc problems 3 errors
        ▼ Greeting 3 errors
              Required Javadoc is absent
              Required Javadoc is absent
              Required Javadoc is absent
▶ Performance issues 1 warning
```