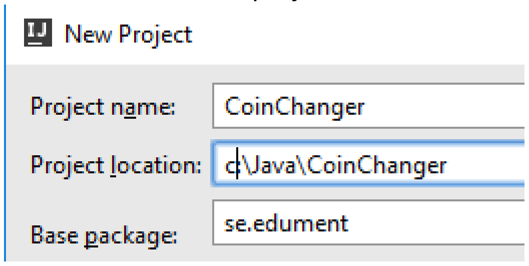
Exercises 3 – Coin changer (Advanced)

Exercise 3.1 – The kata

Main task

In this exercise, you'll get started with TDD by trying out JUnit and writing some simple unit tests. You'll implement a coin changer application. The user calls a method to calculate the change that should be given, optimizing for using the coins with highest value. For instance, given the coin types 1 and 5, the value 23 should return four 5s and three 1s.

1. Create a new IntelliJ project named **CoinChanger** with the base package name **se.edument**



New Project

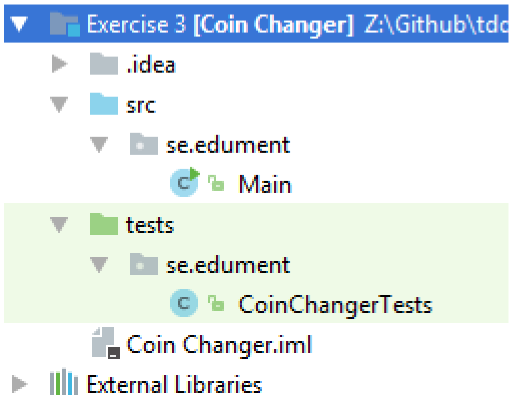
Project name: CoinChanger

Project location: c:\Java\CoinChanger

Base package: se.edument

2. Follow the setup steps in exercise 1 to add the Tests folder and the JUnit library reference.

3. Create a new class for your tests, called **CoinChangerTests** and place it in the **tests.se.edument** folder.



```
Exercise 3 [Coin Changer] Z:\Github\tdc
  .idea
  src
    se.edument
      Main
  tests
    se.edument
      CoinChangerTests
  Coin Changer.iml
External Libraries
```

4. In this new class, add your first test method. Make it **public void** and call it **correctChangeWhenUsingOneCoinType**. It should take no parameters.

5. Put a **@Test** annotation above the method.

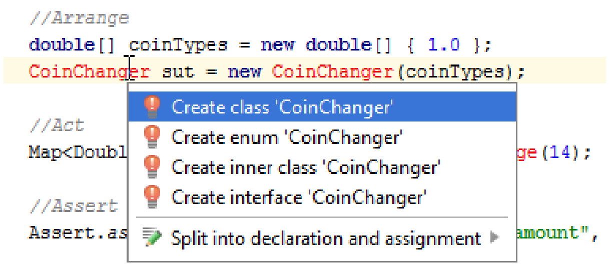
6. Add the following test code:

```
//Arrange
double[] coinTypes = new double[] { 1.0 };
CoinChanger sut = new CoinChanger(coinTypes);

//Act
Map<Double, Integer> myChange = sut.makeChange(14.0);

//Assert
Assert.assertEquals("got the right amount", 14, (int)myChange.get(1.0));
```

Comments:

- We place the result in the myChange variable and we'll use a **Map** as the data type. (You'll need to import **Map** from **java.util.Map**.)
- In the assert part we make sure that we get enough 1's back in change. Since 1 is the only coin type present in the object, we should simply make sure that we've returned 14 of those.
- You'll need to import **org.junit.Assert** for this to work.

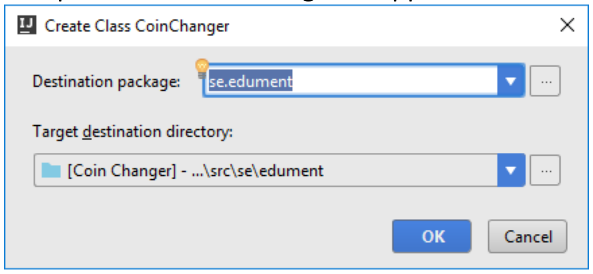
7. Time to create the **CoinChanger** class! Just put the cursor on the **CoinChanger** type and then press **alt+enter** to create the class:



```
//Arrange
double[] coinTypes = new double[] { 1.0 };
CoinChanger sut = new CoinChanger(coinTypes);

//Act
Map<Doubl                                      ge(14);

//Assert
Assert.as                                      amount",
```

Create class 'CoinChanger'
Create enum 'CoinChanger'
Create inner class 'CoinChanger'
Create interface 'CoinChanger'
Split into declaration and assignment

Just press OK in the dialog that appears:



Create Class CoinChanger

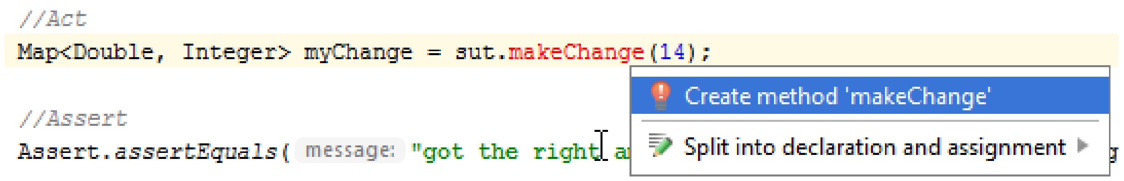
Destination package: se.edument

Target destination directory: [Coin Changer] - ...\src\se\edument

8. Go to the **CoinChanger** class and add this constructor:

```
private double[] coinTypes;

public CoinChanger(double[] coinTypes) {

    this.coinTypes = coinTypes;
}
```

9. Go back to the **CoinChangerTests** class and place the cursor on the **makeChange** method. Then press **Alt+Ente**r on the to generate it:



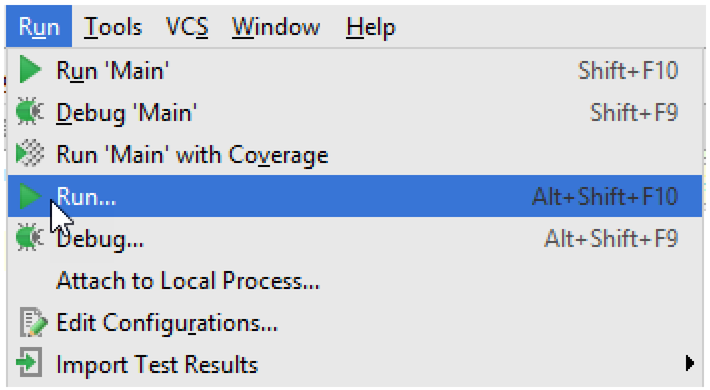
```
//Act
Map<Double, Integer> myChange = sut.makeChange(14);

//Assert
Assert.assertEquals( message: "got the right a
```

Create method 'makeChange'
Split into declaration and assignment

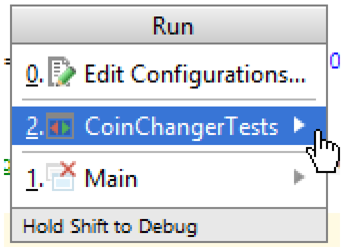
10. Add a **`return null;`** to the method to make sure the code compiles:

```
public Map<Double,Integer> makeChange(double v) {
    return null;
}
```

11. Click on **Run -> Run**



And then run the **CoinChangerTests**:



12. Now run the test and the test should fail.
13. Go back to your code and implement just enough logic in **CoinChanger** to make the test pass. (Hint: a **HashMap** with the contents required by the **assertEquals** should be enough.) Confirm that the test passes by re-running it (play button in the JUnit pane).
14. At this point, think of any refactors you can make to the code.
15. Add another test. This time, make the test about what happens when you pass in two coin types (for instance, 1.0 and 5.0).
16. Make this test pass with the minimal amount of additions and changes.

Stretch task

- What happens if you pass in coin types in a different order? Write a unit test to make sure this still works. If it doesn't pass already, make it pass.
- What happens if the coins in the currency aren't whole numbers? For instance, if you want change for 13.75 and have the currency denominations 0.25, 0.50, 1.00 and 5.00? Write a unit test covering this case.
- We haven't gone through how to handle error conditions yet. But since you've finished everything else, spend some time listing (in a file or on paper) all the things that could go wrong when using the coin changer. Faulty data, wrong arguments, etc. Discuss with the instructor if you want.