Break.{IT}

Home->Tasks->Active

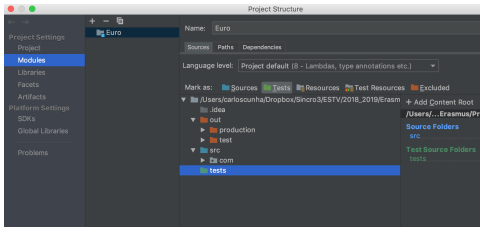
Task: EuroTDD Previous Tasks

Test-driven Development (TDD) exercise

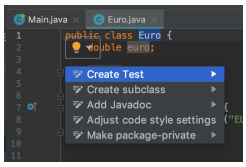
Let's take the code used to demonstrate TDD concepts. Start by **creating the *Euro* class** (*src* folder) with the following code:

```
public class Euro {
double euro;

public Euro(double e) {
   this.euro = e;
}
public String toString(){
   return String.format("EUR %.2f", this.euro);
}

@Override
public boolean equals(Object e2){
   return (e2 instanceof Euro) && this.euro == ((Euro) e2).euro;
}

public Euro minus(Euro oneEuro) {
   return new Euro(this.euro-oneEuro.euro);
}
}
```

Now, we need to create the *EuroTest* class, but before that, we create a folder that will contain all the tests. Afterward, we **mark the folder** as a *Test Source Folder*.



It's possible to use the **quick-fix functionality** in the *Euro* class to create the class *EuroTest*.



Add tests to the **Test Case** *EuroTest*.

```
class EuroTDD {

@BeforeEach
void setUp() {
}

@AfterEach
void tearDown() {
}

@Test
void testEuroObjectCreation(){
   Euro tenEuros = new Euro(10);
}

@Test
void testEuroToString(){
   Euro twoEuros = new Euro(2);
   assertEquals("EUR 2,00", twoEuros.toString());

   twoEuros = new Euro(2.5);
   assertEquals("EUR 2,50", twoEuros.toString());
}

@Test
void testEuroEquality() {
   Euro twoEuros = new Euro(2);
   Euro twoEuros2 = new Euro(2);

   assertTrue(twoEuros.equals(twoEuros2));
}

@Test
void testEuroInequality() {
   Euro twoEuros = new Euro(3);
   Euro sixEuros = new Euro(6);

   assertFalse(twoEuros.equals(sixEuros));
}

@Test
void testEuroEqualsDifferentObject() {
   Euro twoEuros = new Euro(2);
   Double twoEuros2 = new Double(2);

   assertFalse(twoEuros.equals(twoEuros2));
```

```
}

@Test
void testSubtraction() {
    Euro twoEuros = new Euro(2);
    Euro oneEuro = new Euro(1);

    System.out.print(twoEuros.minus(oneEuro));

    assertTrue(new Euro(1).equals(twoEuros.minus(oneEuro)));
}
}
```

Let's continue following the TDD process with the implementation of new features.

Exercise

As it's widely known, representation of *floats* and *doubles* can be inaccurate. To evaluate the numeric safety of class Euro, we can write a test that asserts if **0.61 euros** is equals to **1.03** ***minus*** **0.42**.

What happened? How to fix the problem?

One possible solution is to **store the amount in Cents**. That means that we have to **refactor the amount from double to int**. By **selecting the attribute type** directly in the class, it's possible by **right-clicking the selection** to access the **Type Migration refactoring assistant**. Alternatively, we can change the type in all dependencies.

```
//double euro;
int euro;
```

We may have to **change the values in the test** to **represent cents** instead of euros, **without changing the constructor** to not break dependencies.

Let's run the tests again. **What happened?** We broke the existing functionality, but our tests suite warned us. **Good!**

```
private static final double CENTS_PER_EURO = 100;

//...

return String.format("EUR %.2f", (double) this.euro/CENTS_PER_EURO);
```

Done!!!

Just **create a fake object End()** inside the **@AfterAll method** for the platform to understand when your test case stops running.

```
@AfterAll
static void tearDown() {
  new End();
}
```

Now, you can **submit the JAR for testing**. In case your test *testEuroToString()* has failed, try to determine the root cause of the problem.

Specification

Deadline
2020-04-20 23:55

Submit for Testing | Submit for Evaluation