

ECSE 428 Assignment B
Amees Joshipura (260461226)
Lilly Tong (260459522)

JUnit Test Cases & Screen shots

Test 1

Test name: cToP_returns_double_array

Call setup: cartesianToPolar(3,4)

Expected: cartesianToPolar() returns a double array

Fail:

The screenshot shows an IDE with two tabs: `Calculator.java` and `CalculatorTests.java`. The `Calculator.java` file contains the following code:

```
1 package util;
2
3 public class Calculator {
4
5     public static double[] cartesianToPolar(double x, double y)
6     {
7
8     }
9 }
10
```

The `CalculatorTests.java` file is not visible. The `JUnit` runner is shown at the bottom, indicating the test `cToP_returns_double_array` failed. The failure message is:

```
java.lang.Error: Unresolved compilation problem:
This method must return a result of type double[]

at util.Calculator.cartesianToPolar(Calculator.java:5)
at tests.CalculatorTests.cToP_returns_double_array(CalculatorTests.java:13)
```

The `Outline` view on the right shows the package structure: `util` > `Calculator` > `cartesianToPolar(double, double)`.

Succeed:

The screenshot shows the same IDE with the `Calculator.java` file updated to include a return statement:

```
1 package util;
2
3 public class Calculator {
4
5     public static double[] cartesianToPolar(double x, double y)
6     {
7         double[] result = new double[2];
8         return result;
9     }
10 }
```

The `JUnit` runner now shows the test `cToP_returns_double_array` passed successfully. The `Outline` view on the right shows the package structure: `util` > `Calculator` > `cartesianToPolar(double, double)`.

Test 2

Test name: cToP_inputs_are_zero

Call setup: cartesianToPolar(0,0)

Expected: cartesianToPolar() throws exception with message "One or more inputs are zero!"

Fail:

The screenshot shows an IDE with two tabs: Calculator.java and CalculatorTests.java. The Calculator.java file contains the following code:

```
2  
3 public class Calculator {  
4  
5     public static double[] cartesianToPolar(double x, double y)  
6     {  
7         double[] result = new double[2];  
8         return result; //result[0]=radius, result[1]=angle  
9     }  
10 }  
11
```

The JUnit runner shows the following results:

- Finished after 0.016 seconds
- Runs: 2/2
- Errors: 0
- Failures: 1

The failure trace shows:

- tests.CalculatorTests [Runner: JUnit 4] (0.000 s)
- cToP_inputs_are_zero (0.001 s)
- java.lang.AssertionError
- at tests.CalculatorTests.cToP_inputs_are_zero(CalculatorTests.java:28)

Succeed:

The screenshot shows the same IDE with the Calculator.java file updated to include an exception:

```
4  
5     public static double[] cartesianToPolar(double x, double y) throws Exception  
6     {  
7  
8         if (x==0 || y==0)  
9         {  
10             throw new Exception("One or more inputs are zero!");  
11         }  
12  
13         double[] result = new double[2];  
14     }  
15
```

The JUnit runner shows the following results:

- Finished after 0.011 seconds
- Runs: 2/2
- Errors: 0
- Failures: 0

The test results show:

- tests.CalculatorTests [Runner: JUnit 4] (0.000 s)
- cToP_inputs_are_zero (0.000 s)
- cToP_returns_double_array (0.000 s)

Test 3

Test name: cToP_inputs_are_nonzero

Call setup: cartesianToPolar(3,4)

Expected: cartesianToPolar() returns [5, arctan(4.0/3.0)]

Fail:

The screenshot shows an IDE with two tabs: Calculator.java and CalculatorTests.java. The code in CalculatorTests.java is as follows:

```
5 public static double[] cartesianToPolar(double x, double y) throws Exception
6 {
7     if (x==0 || y==0)
8     {
9         throw new Exception("One or more inputs are zero!");
10    }
11
12    double[] result = new double[2];
13    return result; //result[0]=radius, result[1]=angle
14 }
15
16 }
```

The JUnit test runner shows the following results:

- Finished after 0.023 seconds
- Runs: 3/3
- Errors: 0
- Failures: 1

The test results list shows:

- tests.CalculatorTests [Runner: JUnit 4] (0.000 s)
- cToP_inputs_are_zero (0.000 s)
- cToP_inputs_are_nonzero (0.000 s) - **Failure**
- cToP_returns_double_array (0.000 s)

The failure trace for the cToP_inputs_are_nonzero test is:

- java.lang.AssertionError
- at tests.CalculatorTests.cToP_inputs_are_nonzero(CalculatorTests.java:43)

Succeed:

The screenshot shows the same IDE with the code in CalculatorTests.java updated to calculate the radius and angle:

```
4
5 public static double[] cartesianToPolar(double x, double y) throws Exception
6 {
7     if (x==0 || y==0)
8     {
9         throw new Exception("One or more inputs are zero!");
10    }
11
12    double[] result = new double[2];
13    result[0]=Math.sqrt(Math.pow(x, 2)+Math.pow(y, 2));
14    result[1]=Math.atan(y/x);
15    return result; //result[0]=radius, result[1]=angle
16 }
```

The JUnit test runner shows the following results:

- Finished after 0.013 seconds
- Runs: 3/3
- Errors: 0
- Failures: 0

The test results list shows:

- tests.CalculatorTests [Runner: JUnit 4] (0.001 s)
- cToP_inputs_are_zero (0.001 s)
- cToP_inputs_are_nonzero (0.000 s)
- cToP_returns_double_array (0.000 s)

Test 4

Test name: cToP_contains_negative_input

Call setup: cartesianToPolar(3,-4)

Expected: cartesianToPolar() returns [5, arctan(-4.0/3.0)]

Succeed:

The screenshot shows an IDE with two tabs: Calculator.java and CalculatorTests.java. The Calculator.java file contains the following code:

```
1 package util;
2
3 public class Calculator {
4
5     public static double[] cartesianToPolar(double x, double y) throws Exception
6     {
7
8         if (x==0 || y==0)
9         {
10             throw new Exception("One or more inputs are zero!");
11         }
12
13         double[] result = new double[2];
14         result[0]=Math.sqrt(Math.pow(x, 2)+Math.pow(y, 2));
15         result[1]=Math.atan(y/x);
16         return result; //result[0]=radius, result[1]=angle
17     }
18 }
19
```

The CalculatorTests.java file contains the following code:

```
1 import static org.junit.Assert.*;
2
3 public class CalculatorTests {
4
5     @Test
6     public void cToP_inputs_are_zero() {
7         try {
8             Calculator.cartesianToPolar(0, 0);
9             fail("Expected an exception");
10         } catch (Exception e) {
11             // Expected
12         }
13     }
14
15     @Test
16     public void cToP_inputs_are_nonzero() {
17         double[] result = Calculator.cartesianToPolar(3, 4);
18         assertEquals("radius", 5, result[0], 0.0001);
19         assertEquals("angle", Math.atan(4/3), result[1], 0.0001);
20     }
21
22     @Test
23     public void cToP_returns_double_array() {
24         double[] result = Calculator.cartesianToPolar(3, 4);
25         assertEquals("array length", 2, result.length);
26     }
27
28     @Test
29     public void cToP_contains_negative_input() {
30         double[] result = Calculator.cartesianToPolar(3, -4);
31         assertEquals("radius", 5, result[0], 0.0001);
32         assertEquals("angle", Math.atan(-4/3), result[1], 0.0001);
33     }
34 }

```

The Outline view on the right shows the package structure: util > Calculator > cartesianToPolar(double, double).

The JUnit test results show that all tests passed:

- tests.CalculatorTests [Runner: JUnit 4] (0.001 s)
- cToP_inputs_are_zero (0.001 s)
- cToP_inputs_are_nonzero (0.000 s)
- cToP_returns_double_array (0.000 s)
- cToP_contains_negative_input (0.000 s)

The test runner summary shows: Runs: 4/4, Errors: 0, Failures: 0.

Test 5

Test name: cToP_inputs_too_large

Call setup: cartesianToPolar(10000,-10000)

Expected: cartesianToPolar() throws exception with message "Inputs too large!"

Fail:

The screenshot shows an IDE with two tabs: `Calculator.java` and `CalculatorTests.java`. The `Calculator.java` file contains the following code:

```
1 package util;
2
3 public class Calculator {
4
5     public static double[] cartesianToPolar(double x, double y) throws Exception
6     {
7
8         if (x==0 || y==0)
9         {
10             throw new Exception("One or more inputs are zero!");
11         }
12
13         double[] result = new double[2];
14         result[0]=Math.sqrt(Math.pow(x, 2)+Math.pow(y, 2));
15         result[1]=Math.atan(y/x);
16         return result; //result[0]=radius, result[1]=angle
17     }
18 }
```

The `CalculatorTests.java` file is not visible. The `JUnit` runner shows the test results:

- Finished after 0.024 seconds
- Runs: 5/5
- Errors: 0
- Failures: 1

The test results list shows the following tests:

- cToP_inputs_are_zero (0.000 s)
- cToP_inputs_are_nonzero (0.000 s)
- cToP_inputs_too_large (0.000 s)
- cToP_returns_double_array (0.000 s)
- cToP_contains_negative_input (0.000 s)

The `cToP_inputs_too_large` test is highlighted in red, indicating a failure. The failure trace shows:

```
java.lang.AssertionError
at tests.CalculatorTests.cToP_inputs_too_large(CalculatorTests.java:71)
```

Succeed:

The screenshot shows the same IDE as before, but with the `Calculator.java` file updated to include the new test case. The code is now:

```
1 package util;
2
3 public class Calculator {
4
5     public static double[] cartesianToPolar(double x, double y) throws Exception
6     {
7
8         if (x==0 || y==0)
9         {
10             throw new Exception("One or more inputs are zero!");
11         }
12
13         else if (Math.abs(x)>=10000 || Math.abs(y)>=10000)
14         {
15             throw new Exception("Inputs too large!");
16         }
17
18         double[] result = new double[2];
19     }
```

The `JUnit` runner shows the test results:

- Finished after 0.016 seconds
- Runs: 5/5
- Errors: 0
- Failures: 0

The test results list shows the following tests:

- cToP_inputs_are_zero (0.000 s)
- cToP_inputs_are_nonzero (0.000 s)
- cToP_inputs_too_large (0.000 s)
- cToP_returns_double_array (0.000 s)
- cToP_contains_negative_input (0.000 s)

All tests are now green, indicating success. The `cToP_inputs_too_large` test is highlighted in green.

Test 6

Test name: cToP_inputs_too_small

Call setup: cartesianToPolar(0.01,-0.01)

Expected: cartesianToPolar() throws exception with message "Inputs too small!"

Fail:

The screenshot shows an IDE with two panes. The top pane displays the `Calculator.java` file with the following code:

```
public class Calculator {  
    public static double[] cartesianToPolar(double x, double y) throws Exception  
    {  
        if (x==0 || y==0)  
        {  
            throw new Exception("One or more inputs are zero!");  
        }  
        else if (Math.abs(x)>=10000 || Math.abs(y)>=10000)  
        {  
            throw new Exception("Inputs too large!");  
        }  
        double[] result = new double[2];  
        result[0]=Math.sqrt(Math.pow(x, 2)+Math.pow(y, 2));  
        result[1]=Math.atan(y/x);  
    }  
}
```

The bottom pane shows the JUnit test results. The test `cToP_inputs_too_small` failed with the following error:

```
java.lang.AssertionError  
at tests.CalculatorTests.cToP_inputs_too_small(CalculatorTests.java:85)
```

Succeed:

The screenshot shows an IDE with two panes. The top pane displays the `CalculatorTests.java` file with the following code:

```
@Test  
public void cToP_inputs_too_small() throws Exception  
{  
    boolean thrown = false;  
    double[] result={-1,-1};  
    try {  
        result = Calculator.cartesianToPolar(0.01,0.01);  
    } catch (Exception e) {  
        thrown = true;  
    }  
    Assert.assertTrue(thrown);  
}
```

The bottom pane shows the JUnit test results. All tests passed, including `cToP_inputs_too_small`.

Test 7

Test name: pToC_returns_double_array

Call setup: polarToCartesian(3,0.5)

Expected: polarToCartesian() returns a double array

Fail:

Calculator.java

```
24 double[] result = new double[2];
25 result[0]=Math.sqrt(Math.pow(x, 2)+Math.pow(y, 2));
26 result[1]=Math.atan(y/x);
27 return result; //result[0]=radius, result[1]=angle
28
29 public static double[] polarToCartesian(double r, double theta)
30 {
31 }
32 }
33
34
```

CalculatorTests.java

```
29 public static double[] polarToCartesian(double r, double theta)
30 {
31 }
32 }
33
34
```

JUnit

Finished after 0.034 seconds

Runs: 7/7 Errors: 1 Failures: 0

tests.CalculatorTests [Runner: JUnit 4] (0.000 s)

- cToP_inputs_are_zero (0.000 s)
- cToP_inputs_are_nonzero (0.000 s)
- pToC_returns_double_array (0.000 s)
- cToP_inputs_too_large (0.000 s)
- cToP_inputs_too_small (0.000 s)
- cToP_returns_double_array (0.000 s)
- cToP_contains_negative_input (0.000 s)

Failure Trace

java.lang.Error: Unresolved compilation problem:
This method must return a result of type double[]

at util.Calculator.polarToCartesian(Calculator.java:29)
at tests.CalculatorTests.pToC_returns_double_array(CalculatorTests.java:91)

Succeed:

Calculator.java

```
24 double[] result = new double[2];
25 result[0]=Math.sqrt(Math.pow(x, 2)+Math.pow(y, 2));
26 result[1]=Math.atan(y/x);
27 return result; //result[0]=radius, result[1]=angle
28
29 public static double[] polarToCartesian(double r, double theta)
30 {
31 double[] result = new double[2];
32 return result; //result[0]=x, result[1]=y
33 }
34 }
```

CalculatorTests.java

```
29 public static double[] polarToCartesian(double r, double theta)
30 {
31 double[] result = new double[2];
32 return result; //result[0]=x, result[1]=y
33 }
34 }
```

JUnit

Finished after 0.018 seconds

Runs: 7/7 Errors: 0 Failures: 0

tests.CalculatorTests [Runner: JUnit 4] (0.000 s)

- cToP_inputs_are_zero (0.000 s)
- cToP_inputs_are_nonzero (0.000 s)
- pToC_returns_double_array (0.000 s)
- cToP_inputs_too_large (0.000 s)
- cToP_inputs_too_small (0.000 s)
- cToP_returns_double_array (0.000 s)
- cToP_contains_negative_input (0.000 s)

Failure Trace

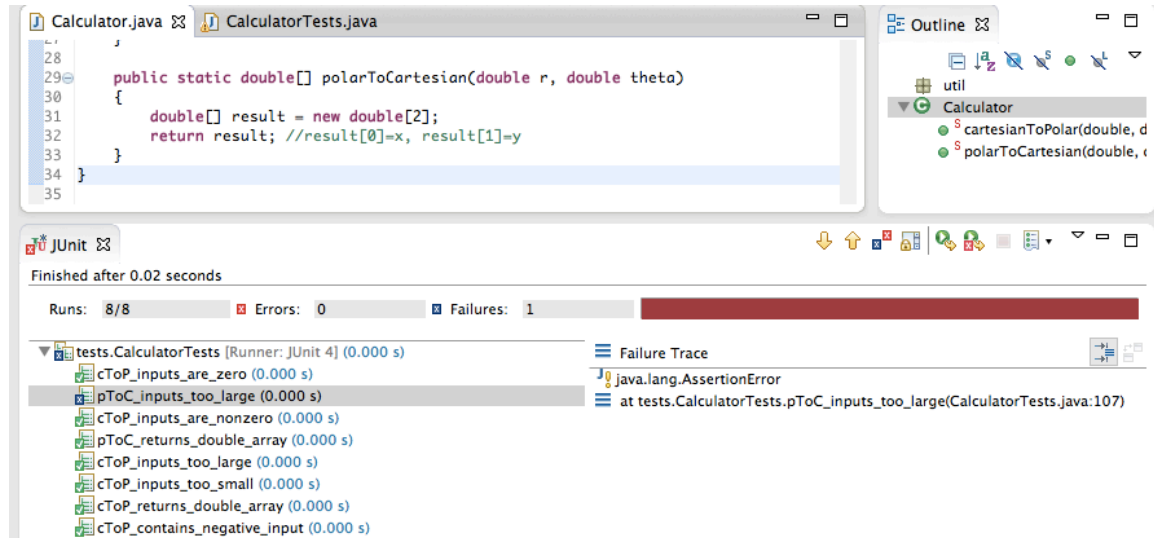
Test 8

Test name: pToC_inputs_too_large

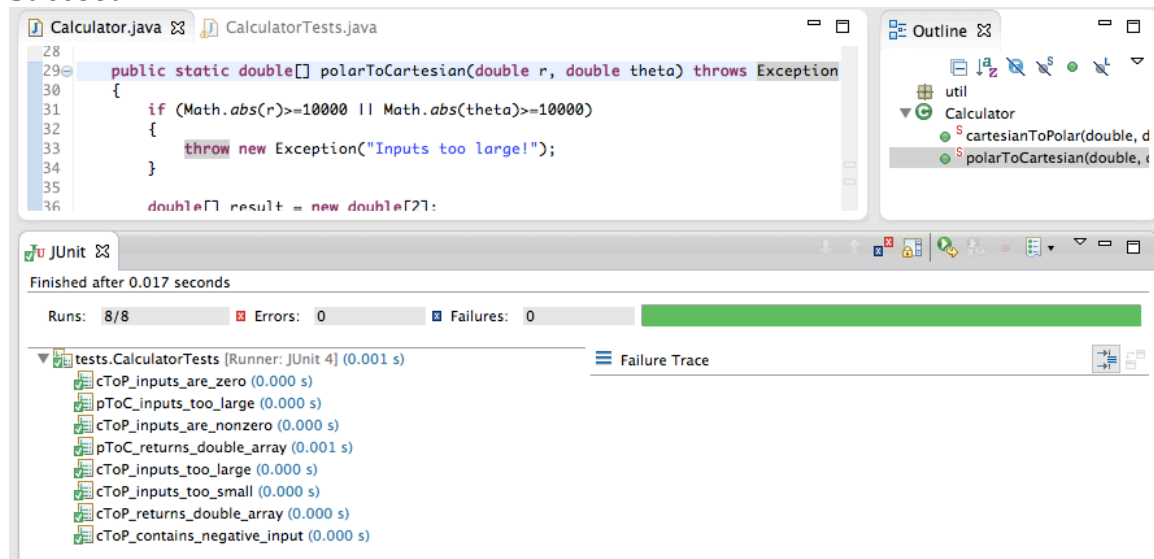
Call setup: polarToCartesian(10000,-10000)

Expected: polarToCartesian () throws exception with message "Inputs too large!"

Fail:



Succeed:



Test 9

Test name: pToC_inputs_are_nonzero

Call setup: polarToCartesian(5,arctan(4/3))

Expected: polarToCartesian() returns [3,4]

Fail:

The screenshot shows an IDE with two tabs: Calculator.java and CalculatorTests.java. The CalculatorTests.java tab is active, showing the test method pToC_inputs_are_nonzero. The test is marked as failed (red icon). The failure trace indicates a java.lang.AssertionError at line 121 of CalculatorTests.java. The test results pane shows that 9 out of 9 runs passed, but 1 failure occurred. The failure trace is as follows:

```
Failure Trace
java.lang.AssertionError
at tests.CalculatorTests.pToC_inputs_are_nonzero(CalculatorTests.java:121)
```

Succeed:

The screenshot shows the same IDE with the CalculatorTests.java tab active. The test method pToC_inputs_are_nonzero is now marked as passed (green icon). The test results pane shows that 9 out of 9 runs passed, with 0 errors and 0 failures. The failure trace is empty.

Test 10

Test name: add2vectors_returns_double_array

Call setup: add2vectors(3,4,5,6)

Expected: add2vectors() returns a double array

Succeed:

The screenshot displays an IDE with two main panels. The top panel shows the source code for `Calculator.java` and `CalculatorTests.java`. The `Calculator` class has a `add2vectors` method that takes four double arguments and returns a double array. The `CalculatorTests` class contains several test methods, including `add2vectors_returns_double_array`. The bottom panel shows the JUnit test runner output, indicating that all tests passed successfully.

```
Calculator.java
38     result[0]=r*Math.cos(theta);
39     result[1]=r*Math.sin(theta);
40     return result; //result[0]=x, result[1]=y
41 }
42
43 public static double[] add2vectors(double x1, double y1, double x2, double y2)
44 {
45     double[] result={0,0};
46     return result;
47 }
48 }
49
```

CalculatorTests.java

```

1 //Test the Calculator class
2
3 import org.junit.Test;
4 import static org.junit.Assert.*;
5
6 public class CalculatorTests {
7
8     @Test
9     public void cToP_inputs_are_zero() {
10         double[] result = Calculator.cToP(0,0);
11         assertEquals("cToP(0,0) should return [0,0]", result);
12     }
13
14     @Test
15     public void pToC_inputs_too_large() {
16         double[] result = Calculator.pToC(1000,1000);
17         assertEquals("pToC(1000,1000) should return [0,0]", result);
18     }
19
20     @Test
21     public void cToP_inputs_are_nonzero() {
22         double[] result = Calculator.cToP(3,4);
23         assertEquals("cToP(3,4) should return [5,6]", result);
24     }
25
26     @Test
27     public void pToC_inputs_are_nonzero() {
28         double[] result = Calculator.pToC(5,6);
29         assertEquals("pToC(5,6) should return [3,4]", result);
30     }
31
32     @Test
33     public void pToC_returns_double_array() {
34         double[] result = Calculator.pToC(5,6);
35         assertTrue("pToC(5,6) should return a double array", result instanceof double[]);
36     }
37
38     @Test
39     public void cToP_inputs_too_large() {
40         double[] result = Calculator.cToP(1000,1000);
41         assertEquals("cToP(1000,1000) should return [0,0]", result);
42     }
43
44     @Test
45     public void cToP_inputs_too_small() {
46         double[] result = Calculator.cToP(0.0001,0.0001);
47         assertEquals("cToP(0.0001,0.0001) should return [0,0]", result);
48     }
49
50     @Test
51     public void cToP_returns_double_array() {
52         double[] result = Calculator.cToP(3,4);
53         assertTrue("cToP(3,4) should return a double array", result instanceof double[]);
54     }
55
56     @Test
57     public void add2vectors_returns_double_array() {
58         double[] result = Calculator.add2vectors(3,4,5,6);
59         assertEquals("add2vectors(3,4,5,6) should return [8,10]", result);
60     }
61
62     @Test
63     public void cToP_contains_negative_input() {
64         double[] result = Calculator.cToP(-3,-4);
65         assertEquals("cToP(-3,-4) should return [-5,-6]", result);
66     }
67
68 }
```

JUnit Runner Output:

Finished after 0.016 seconds

Runs: 10/10 Errors: 0 Failures: 0

tests.CalculatorTests [Runner: JUnit 4] (0.000 s)

- cToP_inputs_are_zero (0.000 s)
- pToC_inputs_too_large (0.000 s)
- cToP_inputs_are_nonzero (0.000 s)
- pToC_inputs_are_nonzero (0.000 s)
- pToC_returns_double_array (0.000 s)
- cToP_inputs_too_large (0.000 s)
- cToP_inputs_too_small (0.000 s)
- cToP_returns_double_array (0.000 s)
- add2vectors_returns_double_array (0.000 s)
- cToP_contains_negative_input (0.000 s)

Test 11

Test name: add2vectors_regular_input

Call setup: add2vectors(3,4,5,6)

Expected: add2vectors() returns [8,10]

Fail:

The screenshot shows an IDE with two tabs: Calculator.java and CalculatorTests.java. The CalculatorTests.java tab is active, showing the add2vectors method. The code is as follows:

```
38     result[0]=r*Math.cos(theta);
39     result[1]=r*Math.sin(theta);
40     return result; //result[0]=x, result[1]=y
41 }
42
43 public static double[] add2vectors(double x1, double y1, double x2, double y2)
44 {
45     double[] result={0,0};
46     return result;
47 }
48 }
49
```

The Outline pane on the right shows the project structure with a 'Calculator' package containing 'cartesianToPolar(double, d)', 'polarToCartesian(double, c)', and 'add2vectors(double, doub'.

The JUnit test runner shows the test 'add2vectors_regular_input' failed. The failure trace indicates a 'java.lang.AssertionError' at 'tests.CalculatorTests.add2vectors_regular_input(CalculatorTests.java:135)'. The test results show 11 runs, 0 errors, and 1 failure.

Succeed:

The screenshot shows the same IDE with the CalculatorTests.java tab active. The add2vectors method has been updated to correctly calculate the sum of two vectors:

```
40     return result; //result[0]=x, result[1]=y
41 }
42
43 public static double[] add2vectors(double x1, double y1, double x2, double y2)
44 {
45     double[] result=new double[2];
46     result[0]=x1+x2;
47     result[1]=y1+y2;
48     return result;
49 }
50 }
51
```

The Outline pane on the right shows the project structure with a 'Calculator' package containing 'cartesianToPolar(double, d)', 'polarToCartesian(double, c)', and 'add2vectors(double, doub'.

The JUnit test runner shows the test 'add2vectors_regular_input' passed. The failure trace is empty. The test results show 11 runs, 0 errors, and 0 failures.

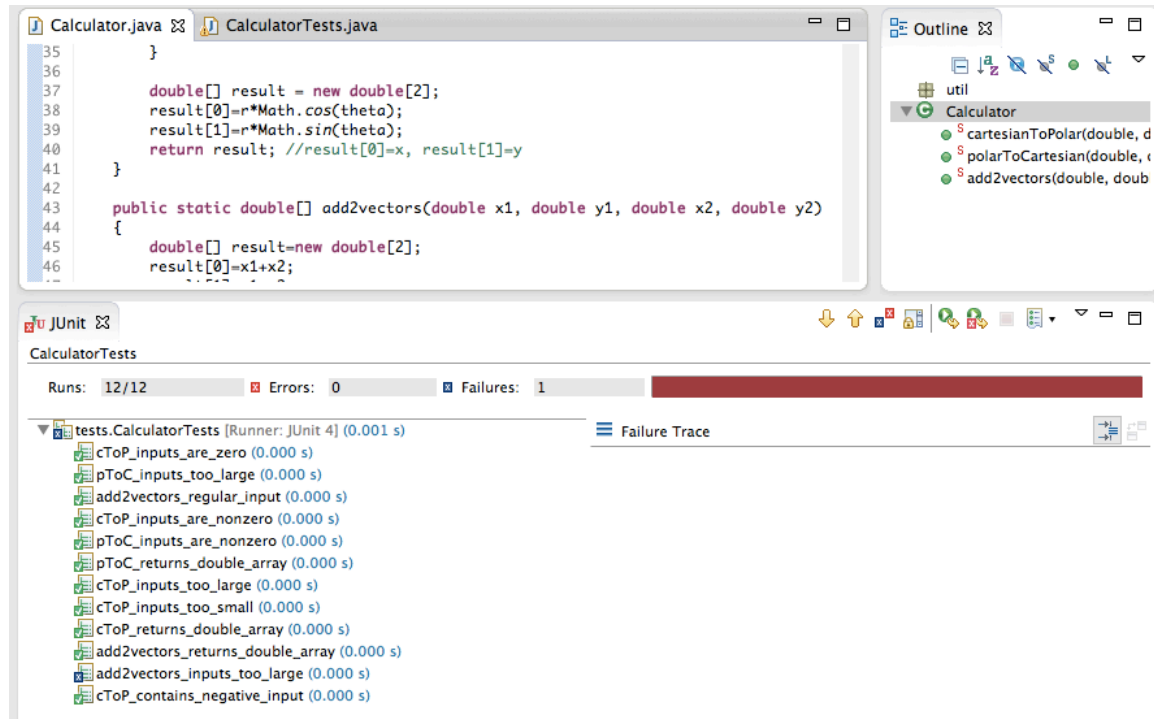
Test 12

Test name: add2vectors_inputs_too_large

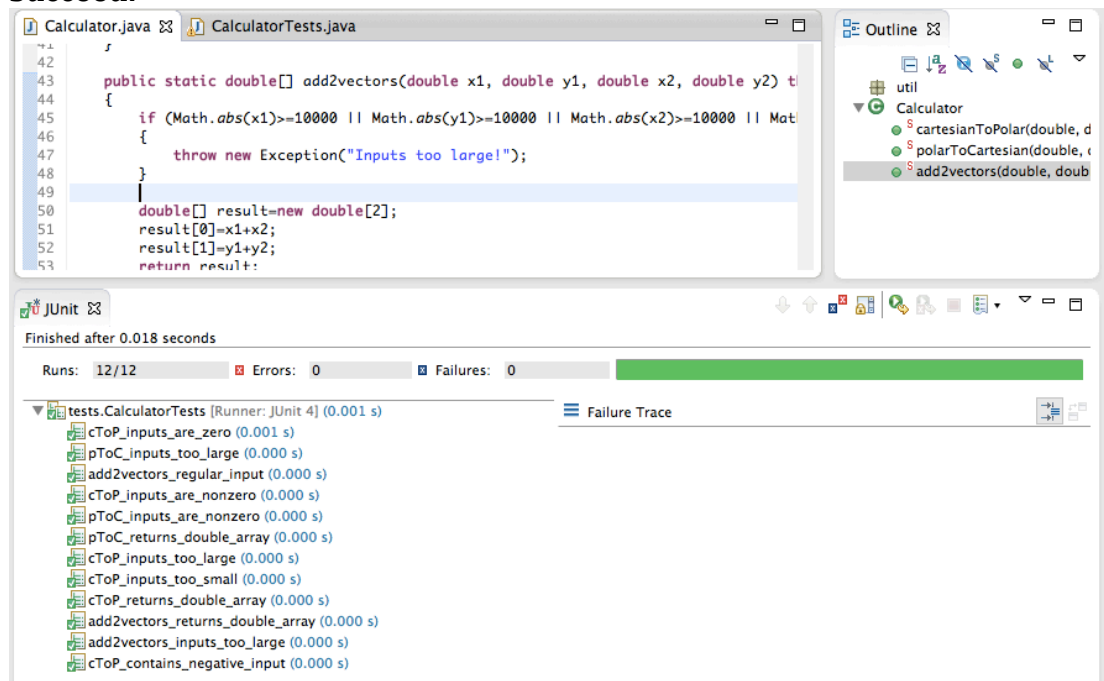
Call setup: add2vectors(10000,-10000,10000,-10000)

Expected: add2vectors() throws exception with message "Inputs too large!"

Fail:



Succeed:



Test 13

Test name: add3vectors_returns_double_array

Call setup: add3vectors(3,4,5,6,7,8)

Expected: add3vectors() returns a double array

Fail:

Calculator.java

```
54 }
55
56 public static double[] add3vectors(double x1, double y1, double x2, double y2, d
57 {
58 |
59 }
60
61 }
```

CalculatorTests.java

```
56 public static double[] add3vectors_returns_double_array() {
57     // Test setup
58     // Test call
59     // Test result
60 }
61 }
```

JUnit

Finished after 0.024 seconds

Runs: 13/13 Errors: 1 Failures: 0

tests.CalculatorTests [Runner: JUnit 4] (0.001 s)

- cToP_inputs_are_zero (0.001 s)
- pToC_inputs_too_large (0.000 s)
- add2vectors_regular_input (0.000 s)
- cToP_inputs_are_nonzero (0.000 s)
- pToC_inputs_are_nonzero (0.000 s)
- pToC_returns_double_array (0.000 s)
- cToP_inputs_too_large (0.000 s)
- cToP_inputs_too_small (0.000 s)
- cToP_returns_double_array (0.000 s)
- add2vectors_returns_double_array (0.000 s)
- add2vectors_inputs_too_large (0.000 s)
- cToP_contains_negative_input (0.000 s)
- add3vectors_returns_double_array (0.000 s)

Failure Trace

java.lang.Error: Unresolved compilation problem:
This method must return a result of type double[]

at util.Calculator.add3vectors(Calculator.java:56)
at tests.CalculatorTests.add3vectors_returns_double_array(CalculatorTests.java:56)

Succeed:

Calculator.java

```
54 }
55
56 public static double[] add3vectors(double x1, double y1, double x2, double y2, d
57 {
58     double[] result = new double[2];
59     result[0] = x1 + x2 + x3;
60     result[1] = y1 + y2 + y3;
61     return result;
62 }
```

CalculatorTests.java

```
56 public static double[] add3vectors_returns_double_array() {
57     // Test setup
58     // Test call
59     // Test result
60 }
61 }
```

JUnit

Finished after 0.021 seconds

Runs: 13/13 Errors: 0 Failures: 0

tests.CalculatorTests [Runner: JUnit 4] (0.000 s)

- cToP_inputs_are_zero (0.000 s)
- pToC_inputs_too_large (0.000 s)
- add2vectors_regular_input (0.000 s)
- cToP_inputs_are_nonzero (0.000 s)
- pToC_inputs_are_nonzero (0.000 s)
- pToC_returns_double_array (0.000 s)
- cToP_inputs_too_large (0.000 s)
- cToP_inputs_too_small (0.000 s)
- cToP_returns_double_array (0.000 s)
- add2vectors_returns_double_array (0.000 s)
- add2vectors_inputs_too_large (0.000 s)
- cToP_contains_negative_input (0.000 s)
- add3vectors_returns_double_array (0.000 s)

Failure Trace

Test 14

Test name: scalarProduct_regular_input

Call setup: scalarProduct(3,4,5,6)

Expected: scalarProduct() returns a 39

Fail:

The screenshot shows an IDE with two tabs: `Calculator.java` and `CalculatorTests.java`. The `Calculator.java` file contains the following code:

```
63  
64 public static double scalarProduct(double x1, double y1, double x2, double y2)  
65 {  
66     return 0;  
67 }  
68  
69  
70 }
```

The `CalculatorTests.java` file is open, showing a list of tests. The test `scalarProduct_regular_input` is highlighted, indicating it failed. The JUnit runner shows 14 runs, 0 errors, and 1 failure. The failure trace for `scalarProduct_regular_input` is shown, indicating a failure at 0.005 s.

Succeed:

The screenshot shows the same IDE as before, but the `Calculator.java` file now contains the following code:

```
63  
64 public static double scalarProduct(double x1, double y1, double x2, double y2)  
65 {  
66     return x1*x2+y1*y2;  
67 }  
68  
69  
70 }
```

The `CalculatorTests.java` file is still open, showing the same list of tests. The test `scalarProduct_regular_input` is now highlighted, indicating it passed. The JUnit runner shows 14 runs, 0 errors, and 0 failures. The failure trace for `scalarProduct_regular_input` is shown, indicating a success at 0.000 s.

Test 15

Test name: vectorProduct_regular_input

Call setup: vectorProduct(3,4,5,6)

Expected: vectorProduct() returns a -2

Fail:

The screenshot shows an IDE with two tabs: Calculator.java and CalculatorTests.java. The Calculator.java file contains the following code:

```
68
69 public static double vectorProduct(double x1, double x2, double y1, double y2)
70 {
71     return 0; //this is just the value of the z component! so answer will be (0,
72 }
73
74
75 }
```

The test results pane shows the following tests:

- cToP_inputs_are_zero (0.001 s)
- pToC_inputs_too_large (0.000 s)
- add2vectors_regular_input (0.000 s)
- cToP_inputs_are_nonzero (0.000 s)
- pToC_inputs_are_nonzero (0.000 s)
- vectorProduct_regular_input (0.000 s)**
- pToC_returns_double_array (0.000 s)
- cToP_inputs_too_large (0.000 s)
- cToP_inputs_too_small (0.001 s)
- cToP_returns_double_array (0.000 s)
- add2vectors_returns_double_array (0.000 s)
- add2vectors_inputs_too_large (0.000 s)
- cToP_contains_negative_input (0.000 s)
- add3vectors_returns_double_array (0.000 s)
- scalarProduct_regular_input (0.000 s)

The failure trace shows:

```
java.lang.AssertionError
at tests.CalculatorTests.vectorProduct_regular_input(CalculatorTests.java:17)
```

Succeed:

The screenshot shows an IDE with two tabs: Calculator.java and CalculatorTests.java. The Calculator.java file contains the following code:

```
68
69 public static double vectorProduct(double x1, double x2, double y1, double y2)
70 {
71     return x1*y2-y1*x2; //this is just the value of the z component! so answer w
72 }
73
74
75 }
```

The test results pane shows the following tests:

- cToP_inputs_are_zero (0.000 s)
- pToC_inputs_too_large (0.000 s)
- add2vectors_regular_input (0.000 s)
- cToP_inputs_are_nonzero (0.000 s)
- pToC_inputs_are_nonzero (0.000 s)
- vectorProduct_regular_input (0.000 s)
- pToC_returns_double_array (0.000 s)
- cToP_inputs_too_large (0.000 s)
- cToP_inputs_too_small (0.000 s)
- cToP_returns_double_array (0.000 s)
- add2vectors_returns_double_array (0.000 s)
- add2vectors_inputs_too_large (0.000 s)
- cToP_contains_negative_input (0.000 s)
- add3vectors_returns_double_array (0.001 s)
- scalarProduct_regular_input (0.000 s)