More With Unit Testing

Brandon Krakowsky

Penn Engineering

Sentence Counter Project

Sentence Counter

- Create a new *SentenceCounter* program. Write a simple *countSentences* method that counts the number of sentences in a string of text, based on a period (.)
- For now, just create the method stub and return 0
    - Again, the idea here is to start with an empty method so our tests fail

Sentence Counter

- Create a new *SentenceCounter* program. Write a simple *countSentences* method that counts the number of sentences in a string of text, based on a period (.)
- For now, just create the method stub and return 0
  - Again, the idea here is to start with an empty method so our tests fail

SentenceCounter.java

```
1  package sentences;
2
3  public class SentenceCounter {
4
5      /**
6       * Counts the number of sentences in the given text.
7       * @param text to extract number of sentences
8       * @return count of sentences
9       */
10     int countSentences(String text) {
11         return 0;
12     }
13 }
14
```

Sentence Counter Testing

- Create a new testing file and create your tests.
  - All tests should fail

```
 9    @Test
10    void testCountSentences() {
11
12        SentenceCounter sc = new SentenceCounter();
13
14        assertEquals(2, sc.countSentences("Today is Monday. This is it."));
15        assertEquals(1, sc.countSentences("."));
16        assertEquals(2, sc.countSentences(".."));
17        assertEquals(2, sc.countSentences("Hi there. My name is Brandon. What's yours?"));
18        assertEquals(0, sc.countSentences(""));
19
20    }
21
```

Sentence Counter Testing

- Implement the *countSentences* method to pass the tests
  - It can count the number of sentences, based on a period (.)

```
7       /**
8        * Counts the number of sentences in the given text.
9        * @param text to extract number of sentences
10       * @return count of sentences
11       */
12      int countSentences(String text) {
13
14          //strip whitespace from beginning and end of string
15          text = text.strip();
16
17          int puncCount = 0;
18          //iterate over string and find instances of period (.)
19          for (int i = 0; i < text.length(); i++) {
20              if (text.charAt(i) == '.') {
21                  puncCount++;
22              }
23          }
24
25          int sentenceCount = puncCount;
26          return sentenceCount;
27      }
```

Sentence Counter Testing

- All tests should pass, including the test with a question
  - The last sentence in the text is followed by a question mark (?), so we would only expect to get a sentence count of 2, based on a period (.)

```
 9    @Test
10    void testCountSentences() {
11
12        SentenceCounter sc = new SentenceCounter();
13
14        assertEquals(2, sc.countSentences("Today is Monday. This is it."));
15        assertEquals(1, sc.countSentences("."));
16        assertEquals(2, sc.countSentences(".."));
17        assertEquals(2, sc.countSentences("Hi there. My name is Brandon. What's yours?"));
18        assertEquals(0, sc.countSentences(""));
19
20    }
21
```

Sentence Counter Testing

- But we SHOULD account for other kinds of punctuation, like a question mark (?)
  - Let's fix it. First, update the test with the new expected value (3)
  - If you run this test, it SHOULD fail!

```
 9    @Test
10    void testCountSentences() {
11
12        SentenceCounter sc = new SentenceCounter();
13
14        assertEquals(2, sc.countSentences("Today is Monday. This is it."));
15        assertEquals(1, sc.countSentences("."));
16        assertEquals(2, sc.countSentences(".."));
17        assertEquals(3, sc.countSentences("Hi there. My name is Brandon. What's yours?"));
18        assertEquals(0, sc.countSentences(""));
19
20    }
```

Sentence Counter Testing

- Update the *countSentences* method to account for a question mark (?) and to pass the test

```
5      /**
6       * Counts the number of sentences in the given text.
7       * @param text to extract number of sentences
8       * @return count of sentences
9       */
10     int countSentences(String text) {
11
12         //strip whitespace from beginning and end of string
13         text = text.strip();
14
15         int puncCount = 0;
16         //iterate over string and find instances of period (.)
17         for (int i = 0; i < text.length(); i++) {
18             if (text.charAt(i) == '.' || text.charAt(i) == '?') {
19                 puncCount++;
20             }
21         }
22
23         int sentenceCount = puncCount;
24         return sentenceCount;
25     }
```

Optional setUp & tearDown Methods

- The following are optional methods (and annotations) when running a testing file
  - These are typically defined at the top of your testing class

```
@BeforeEach
void setUp()
```

  - The setUp method (annotated by @BeforeEach) runs before each unit test method
  - Can be used to initialize everything to a "clean" state

```
@AfterEach
void tearDown()
```

  - The tearDown method (annotated by @AfterEach) runs after each unit test method
  - Can be used to remove artifacts (such as files) that may have been created

Counter Project

Counter Class

- Let's create a "Counter" class
  - The class will declare a counter (int) and initialize it to zero
  - The *increment* method will add one to the counter and return the new value
  - The *decrement* method will subtract one from the counter and return the new value

Create Counter Class

Counter.java

```
1
2  /**
3   * Represents a count, with methods.
4   * @author lbrandon
5   *
6   */
7  public class Counter {
8
9      //instance variable(s)
10
11     /**
12      * Stores internal count.
13      * Primitive int, defaults to 0.
14      */
15     int count;
16
```

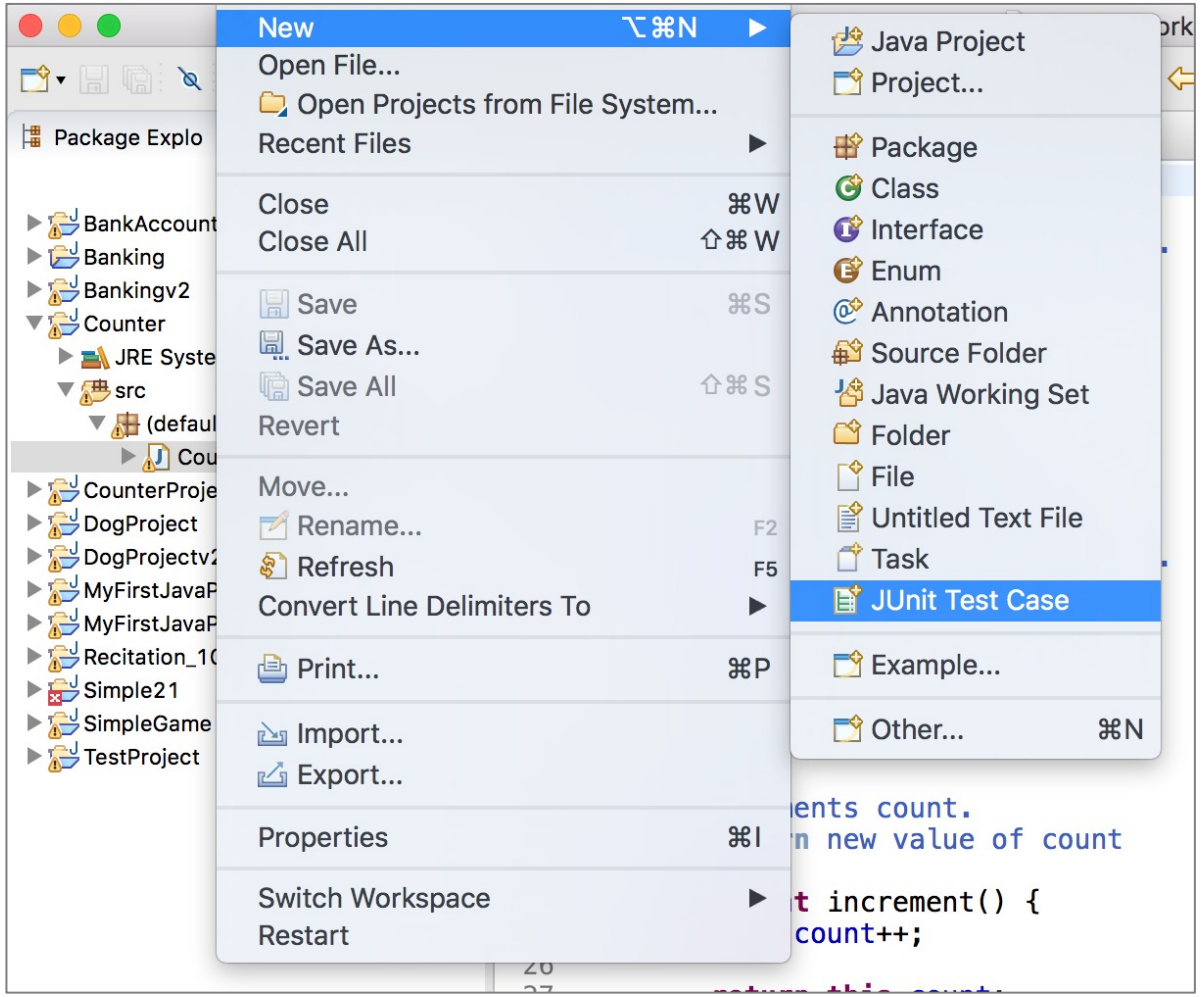
Create Counter Class

```
17
18      //methods
19
20      /**
21       * Increments count.
22       * @return new value of count
23       */
24      public int increment() {
25          this.count++;
26
27          return this.count;
28      }
29
30
31      /**
32       * Decrement count.
33       * @return new value of count
34       */
35      public int decrement() {
36          this.count--;
37
38          return this.count;
39      }
40
41      /**
42       * Returns current value of count.
43       * @return count
44       */
45      public int getCount() {
46          return this.count;
47      }
48
```

- Is JUnit testing overkill for these methods?
  - Doesn't matter, writing JUnit tests for trivial classes is no big deal
- Note: Often, you won't write tests for simple "getter" methods like *getCount*

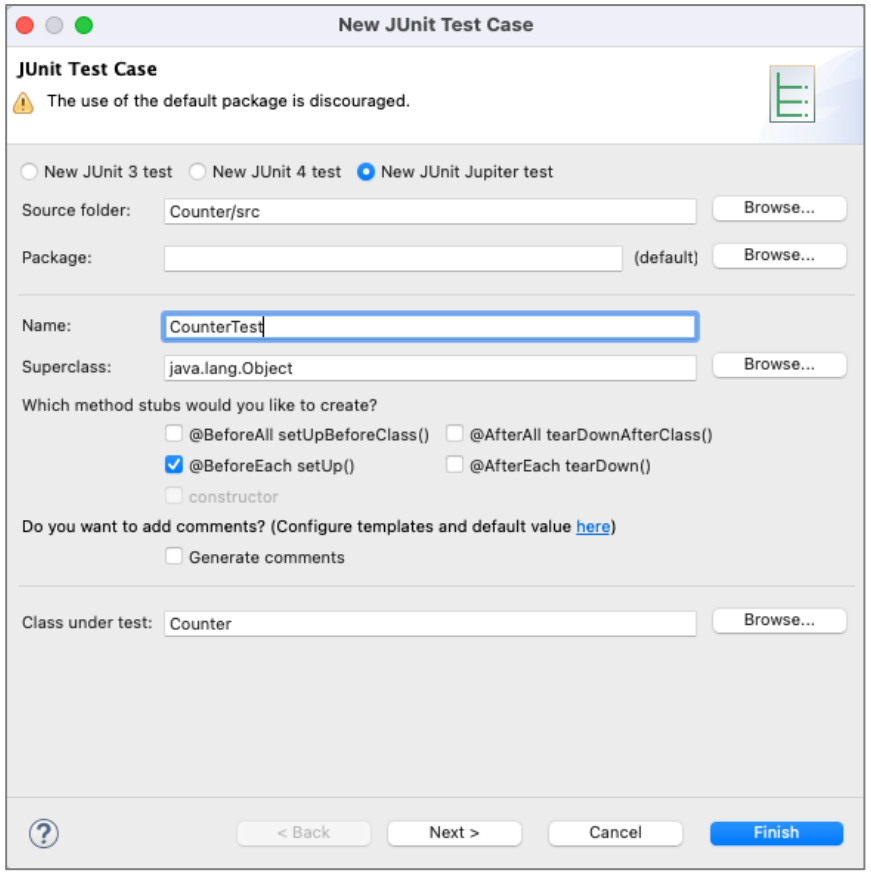
Create JUnit Tests

- Select the class file in the Package Explorer, and go to “New” → “JUnit Test Case”



Create JUnit Tests

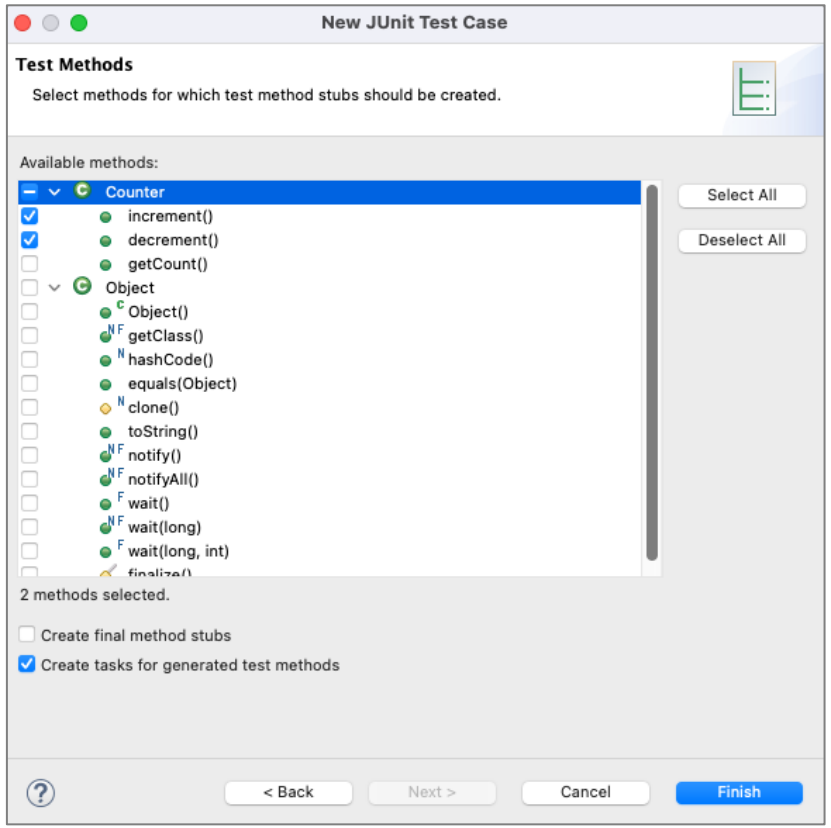
- Use the default name provided for your JUnit Test Case class



Make sure @New Junit Jupiter test is selected
Also make sure @BeforeEach setUp() is checked

Create JUnit Tests

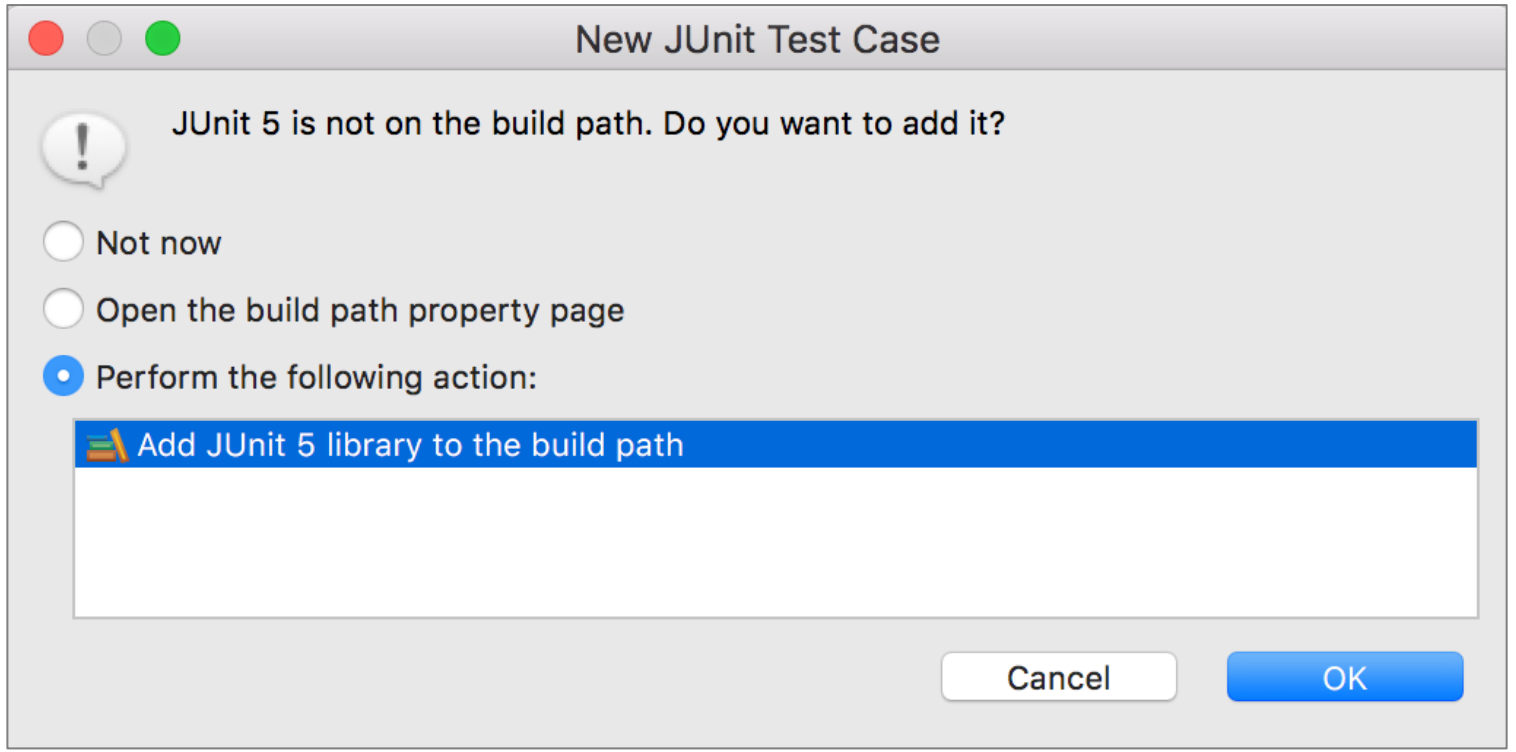
- To have Eclipse generate test method stubs for you, use the checkboxes to decide which methods you want test cases for. Don't select Object or anything under it.



Check Create tasks for generated test methods

Create JUnit Tests

- If not added already, add the JUnit 5 library to the project build path
  - This includes the necessary JUnit framework in your project



Create JUnit Tests

- Eclipse will add a new JUnit Test class in the same package (or default)
  - You'll see test method stubs to be implemented
  - The code in each test method is calling *fail* (with a message), to force the test methods to initially fail

Counter.java | CounterTest.java

```
1  import static org.junit.jupiter.api.Assertions.*;
5
6  class CounterTest {
7
8      @BeforeEach
9      void setUp() throws Exception {
10     }
11
12     @Test
13     void testIncrement() {
14         fail("Not yet implemented"); // TODO
15     }
16
17     @Test
18     void testDecrement() {
19         fail("Not yet implemented"); // TODO
20     }
21
22 }
23
```

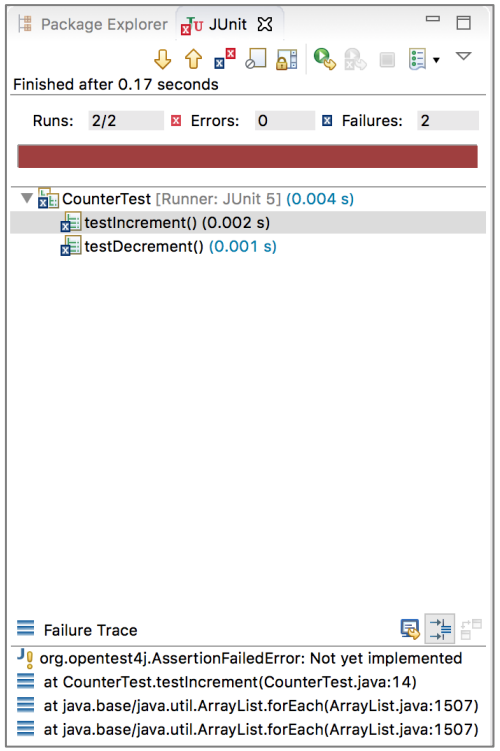
The *setUp* method (annotated by @BeforeEach) runs before each unit test method

Each test method is annotated by @Test

Note: You can't be concerned with the order in which unit test methods run

Create JUnit Tests

- If you run the tests, they should ALL fail
  - Eclipse will open the JUnit panel (on the left)
    - The top bar will show red
    - The number next to “Failures” will show 2
    - The message at the bottom will explain why the tests failed



Create JUnit Tests

- Declare and initialize a Counter object for testing

Counter.java | CounterTest.java

```
1  import static org.junit.jupiter.api.Assertions.*;
2
3  import org.junit.jupiter.api.BeforeEach;
4  import org.junit.jupiter.api.Test;
5
6  class CounterTest {
7
8      Counter counter; //Declare a Counter for testing
9
10     //runs before each unit test method
11     //not required
12     @BeforeEach
13     void setUp() throws Exception {
14         this.counter = new Counter(); //Initialize the Counter here
15     }
16
```

Create JUnit Tests

- Implement the test methods, adding test cases with assert methods

```
16
17      @Test
18      void testIncrement() {
19
20          //asserts that calling increment returns 1
21          assertTrue(this.counter.increment() == 1);
22
23          //asserts that calling increment returns 2
24          assertTrue(this.counter.increment() == 2);
25
26          //increments again
27          this.counter.increment();
28
29          //asserts that calling increment again does not return 2
30          assertFalse(this.counter.getCount() == 2, "should not return 2 after incrementing again");
31
32          //asserts that 3 is equal to the new count
33          assertEquals(3, this.counter.getCount());
34
35          //asserts that 3 is not equal to calling increment again
36          assertNotEquals(3, this.counter.increment());
37
38      }
39
```

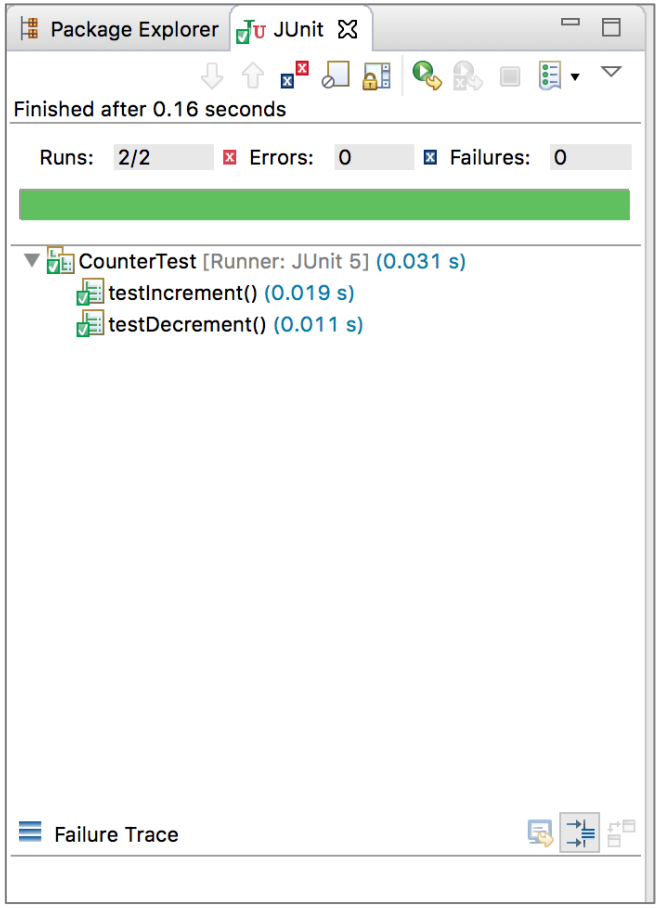
Create JUnit Tests

- Implement the test methods, adding test cases with assert methods

```
39
40      @Test
41      void testDecrement() {
42          //asserts that -1 is equal to calling decrement
43          assertEquals(-1, this.counter.decrement());
44
45          //asserts that calling decrement again returns -2
46          assertTrue(this.counter.decrement() == -2);
47
48          //decrements again
49          this.counter.decrement();
50
51          //asserts that calling decrement again does not return -2
52          assertFalse(this.counter.getCount() == -2, "should not return -2 after decrementing again");
53
54          //asserts that -3 is equal to the new count
55          assertTrue(this.counter.getCount() == -3);
56
57      }
58
```

Create JUnit Tests

- If you run the tests, they should ALL pass
  - In the JUnit panel (on the left) – the top bar will show green
  - The number next to “Failures” will show 0




```java
import static org.junit.jupiter.api.Assertions.*;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

class CounterTest {

    Counter counter; //Declare a Counter for testing

    //runs before each unit test method
    //not required
    @BeforeEach
    void setUp() throws Exception {
        this.counter = new Counter(); //Initialize the Counter here
    }

    @Test
    void testIncrement() {

        //asserts that calling increment returns 1
        assertTrue(this.counter.increment() == 1);

        //asserts that calling increment returns 2
        assertTrue(this.counter.increment() == 2);

        //increments again
        this.counter.increment();

        //asserts that calling increment again does not return 2
        assertFalse(this.counter.getCount() == 2, "should not return 2 after incrementing again");

        //asserts that 3 is equal to the new count
        assertEquals(3, this.counter.getCount());

        //asserts that 3 is not equal to calling increment again
        assertNotEquals(3, this.counter.increment());

    }

    @Test
    void testDecrement() {
        //asserts that -1 is equal to calling decrement
        assertEquals(-1, this.counter.decrement());

        //asserts that calling decrement again returns -2
        assertTrue(this.counter.decrement() == -2);

        //decrements again
        this.counter.decrement();

        //asserts that calling decrement again does not return -2
        assertFalse(this.counter.getCount() == -2, "should not return -2 after decrementing again");

        //asserts that -3 is equal to the new count
        assertTrue(this.counter.getCount() == -3);

    }
```

More Assert Methods

```
void assertThrows(Exception.class, () -> {
    //code that throws an exception
});
```

- Asserts that the enclosed code throws an Exception of a particular type

```
void assertDoesNotThrow(() -> {
    //code that does not throw an exception
});
```

- Asserts that the enclosed code does not throw an Exception

More Assert Methods

```
void assertThrows(Exception.class, () -> {
    //code that throws an exception
});
```

- Asserts that the enclosed code throws an Exception of a particular type

```
void assertDoesNotThrow(() -> {
    //code that does not throw an exception
});
```

- Asserts that the enclosed code does not throw an Exception

- For example:

```
String test = null;
assertThrows(NullPointerException.class, () -> {
    test.length();
});
```

  - Asserts that *test.length()* throws a NullPointerException
  - Why? *test* is null, so there is no method *length()*

Banking Project w/ Unit Testing

Banking Project w/ Unit Testing

- We'll unit test our previous "Banking" project, which had 3 classes
  - Bank
    - Includes the public static void main(String[] args) method
    - No updates needed
  - Customer
    - No updates needed
  - BankAccount
    - Updates needed!
- Create new unit testing classes
  - CustomerTest
    - For testing the Customer class
  - BankAccountTest
    - For testing the BankAccount class

Updated BankAccount Class

- Add a fastCashAmount instance variable

Bank.java | BankAccount.java

```
1  package banking;
2
3  /**
4   * Represents a checking/savings bank account for a customer.
5   * @author lbrandon
6   *
7   */
8  public class BankAccount {
9
10     //instance variables
11
12     /**
13      * Type of account (checking/savings).
14      */
15     String accountType;
16
17     /**
18      * Account balance.
19      */
20     double balance;
21
22     /**
23      * Customer for account.
24      */
25     Customer customer;
26
27     /**
28      * Fast cash for quick withdrawal.
29      */
30     double fastCashAmount;
31
32
```

Updated BankAccount Class

- Update the constructor to set the initial value for fastCashAmount

```
32
33      //constructor
34
35      /**
36       * Creates a bank account of given type for given customer.
37       * Sets default fast cash amount.
38       * @param accountType for bank account
39       * @param customer for this account
40       */
41      public BankAccount(String accountType, Customer customer) {
42          this.accountType = accountType;
43          this.customer = customer;
44
45          //set default value for fast cash
46          this.fastCashAmount = 60;
47      }
```

Updated BankAccount Class

- Add the fastWithDraw and setFastCashAmount methods

```
72
73      /**
74       * Withdraws the fast cash amount.
75       * @throws Exception if amount is greater than available balance
76       */
77      public void fastWithDraw() throws Exception {
78          this.withdraw(this.fastCashAmount);
79      }
80
81      /**
82      * Sets the fast cash amount, if the amount is greater than 0.
83      * @param amount to set as fast cash
84      */
85      public void setFastCashAmount(double amount){
86          if(amount > 0){
87              this.fastCashAmount = amount;
88          }
89      }
```

Updated BankAccount Class

- Update the deposit method

```
51      /**
52       * Deposits the given balance, if the balance is greater than 0.
53       * @param balance to add
54       */
55      public void deposit(double balance) {
56          if (balance > 0) {
57              this.balance += balance;
58          }
59      }
60
```

Create CustomerTest Class

Bank.java | BankAccount.java | Customer.java | CustomerTest.java

```java
package banking;

import static org.junit.jupiter.api.Assertions.*;

class CustomerTest {

    //Declare Customer for testing
    Customer customer;

    @BeforeEach
    void setUp() throws Exception {
        //Initialize Customer here
        this.customer = new Customer("Brandon");
    }

    @Test
    void testSetAddress() {

        //Get expected address, should be null to start
        assertNull(this.customer.getAddress());

        //Set new address
        this.customer.setAddress("Brooklyn, NY");

        //Get expected address
        assertEquals("Brooklyn, NY", this.customer.getAddress());

        //Set new address
        this.customer.setAddress("Cranston, RI");

        //Get expected address
        assertEquals("Cranston, RI", this.customer.getAddress());
    }

}
```

Create BankAccountTest Class

Bank.java | BankAccount.java | BankAccountTest.java

```
1  package banking;
2  import static org.junit.jupiter.api.Assertions.*;
7
8  class BankAccountTest {
9
10     //Declare Customer and BankAccounts for testing
11     Customer customer;
12     BankAccount myCheckingAccount;
13     BankAccount mySavingsAccount;
14
15     //runs before each test method
16     @BeforeEach
17     void setUp() throws Exception {
18
19         //Initialize Customer and BankAccounts here
20         this.customer = new Customer("Brandon");
21         this.myCheckingAccount = new BankAccount("checking", this.customer);
22         this.mySavingsAccount = new BankAccount("savings", this.customer);
23     }
24
```

Create BankAccountTest Class

```
24      @Test
25      void testDeposit() {
26
27          //make deposit
28          this.myCheckingAccount.deposit(100);
29
30          //test that current balance is 100.00
31          assertEquals(100.00, this.myCheckingAccount.balance, 0.000001);
32
33          //make deposit of negative amount
34          this.myCheckingAccount.deposit(-100);
35
36          //balance should be the same
37          assertEquals(100.00, this.myCheckingAccount.balance, 0.000001);
38
39          //deposit 0
40          this.myCheckingAccount.deposit(0);
41
42          //balance should be the same
43          assertEquals(100.00, this.myCheckingAccount.balance, 0.000001);
44      }
45
```

Create BankAccountTest Class

```
46      @Test
47      void testWithdraw() {
48
49          //deposit initial 100 into savings
50          this.mySavingsAccount.deposit(100);
51
52          //assert current balance
53          assertEquals(100.00, this.mySavingsAccount.balance, 0.000001);
54
55          //try to make withdrawal of 80
56          try {
57              this.mySavingsAccount.withdraw(80);
58          } catch (Exception e) {
59              // TODO Auto-generated catch block
60              e.printStackTrace();
61          }
62
63          //balance should be 20.00
64          assertEquals(20.00, this.mySavingsAccount.balance, 0.000001);
65
```

Create BankAccountTest Class

```
66          //try to withdraw amount greater than available balance
67          //expect an exception
68          assertThrows(Exception.class, () -> {
69              this.mySavingsAccount.withdraw(21);
70          });
71
72          //balance should be the same
73          assertEquals(20.00, this.mySavingsAccount.balance, 0.000001);
74
75          //try to make withdrawal
76          //does not throw exception
77          assertDoesNotThrow(() -> {
78              this.mySavingsAccount.withdraw(19);
79          });
80
81          //balance should be 1.00
82          assertEquals(1.00, this.mySavingsAccount.balance, 0.000001);
83
84      }
85
```

Create BankAccountTest Class

```
86      @Test
87      void testFastWithdraw() {
88
89          //initial deposit
90          this.myCheckingAccount.deposit(100);
91
92          //try to make fast withdrawal
93          //default amount is 60
94          try {
95              this.myCheckingAccount.fastWithdraw();
96          } catch (Exception e) {
97              // TODO Auto-generated catch block
98              e.printStackTrace();
99          }
100
101         //balance should be 40.00
102         assertEquals(40.00, this.myCheckingAccount.balance, 0.000001);
103
```

Create BankAccountTest Class

```
104         //reset the fast cash amount to 20
105         this.myCheckingAccount.setFastCashAmount(20);
106
107         //assert that an exception is not thrown when I make fast cash withdrawal
108         assertDoesNotThrow(() -> {
109             this.myCheckingAccount.fastWithdraw();
110         });
111
112         //check balance
113         assertEquals(20.00, this.myCheckingAccount.balance, 0.000001);
114
115         //set fast cash amount to negative value
116         //should be ignored
117         this.myCheckingAccount.setFastCashAmount(-50);
118
119         //should still withdraw 20
120         try {
121             this.myCheckingAccount.fastWithdraw();
122         } catch (Exception e) {
123             // TODO Auto-generated catch block
124             e.printStackTrace();
125         }
126
127         //balance should be 0 after fast withdrawal
128         assertEquals(0.00, this.myCheckingAccount.balance, 0.000001);
129
```

Create BankAccountTest Class

```
130         //expecting exception when I try to make fast withdrawal because of 0 balance
131         assertThrows(Exception.class, () -> {
132             this.myCheckingAccount.fastWithdraw();
133         });
134
135     }
136
```