**Test Driven Development :**

The following steps were taken :

1. Identify a suitable class to work from
2. Identify the classes/methods to be coded
3. Create a ‘stub’ set of JUnit tests
4. Run the stub tests – all will fail
5. Create a ‘stub’ set of methods
6. Update all JUnit tests to fail on a ‘false’ return
7. Run the tests – all will fail
8. Loop through the test cycle (JUnit and methods one by one)
   1. Write initial ‘full’ JUnit test
   2. Run JUnit tests – fail
   3. Write the class method
   4. Run JUnit test – pass

**Class to test**

The class UserMarker is used to store information about a device and its current location. It was coded with a Builder Factory so that an instance of the class has to be created with data.

|  |
| --- |
| //081113 - MtpA - Refactor to change getDeviceID return variable to String  //311013 - MtpA - Changed ID data from Integer to String to reflect XML changes  // Changed variable name deviceID to deviceID to be more reflective of actual meaning  //221013 - MtpA - Create class with details of the current position of the app  package com.liftme.liftmeclient;  import java.text.DateFormat;  import java.text.SimpleDateFormat;  import java.util.Calendar;  public class UserMarker {  private final String deviceID;  private final double markerLat;  private final double markerLong;  private final Calendar markerCal;  private final DateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");  private final String markerDate;  private final String markerTime;  public static class Builder {  // Required parameters  private final String deviceID;  private double markerLat;  private double markerLong;  private Calendar markerCal;  private DateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");  private String markerDate;  private String markerTime;  // Optional parameters  public Builder(String deviceID, double markerLat, double markerLong, Calendar markerCal) {  this.deviceID = deviceID;  this.markerLat = markerLat;  this.markerLong = markerLong;  this.markerCal = markerCal;  this.markerDate = dateFormat.format(markerCal.getTime()).substring(0, 10);  this.markerTime = dateFormat.format(markerCal.getTime()).substring(11);  } // Builder constructor - no args  public UserMarker build() {  return new UserMarker(this);  } // method build  } // class Builder  private UserMarker(Builder builder) {  deviceID = builder.deviceID;  markerLat = builder.markerLat;  markerLong = builder.markerLong;  markerCal = builder.markerCal;  markerDate = builder.markerDate;  markerTime = builder.markerTime;  } // builder constructor  public String getDeviceID() {  return deviceID;  }  public double getMarkerLat() {  return markerLat;  }  public double getMarkerLong() {  return markerLong;  }  public Calendar getMarkerCal() {  return markerCal;  }  public DateFormat getDateFormat() {  return dateFormat;  }  public String getMarkerDate() {  return markerDate;  }  public String getMarkerTime() {  return markerTime;  }    } // class userMarker |

However, it can be seen that although data has to be passed to the constructor there is no validation on the arguments.

Therefore a data validator class is needed with the JUnit test code.

**Classes/Methods to be coded :**

* LiftMeDataValidation Class

All methods return a *boolean* argument

* isDeviceIDNotNull – ensure that the DeviceID has data in it
* isLatValid – ensure that the latitude is between -90 and 90
* is LongValid – ensure that the longditude is between -180 and 180
* is DateValid – ensure that a date is valid
* TestLiftMeDataValidation Class

JUnit test methods (initial stubs)

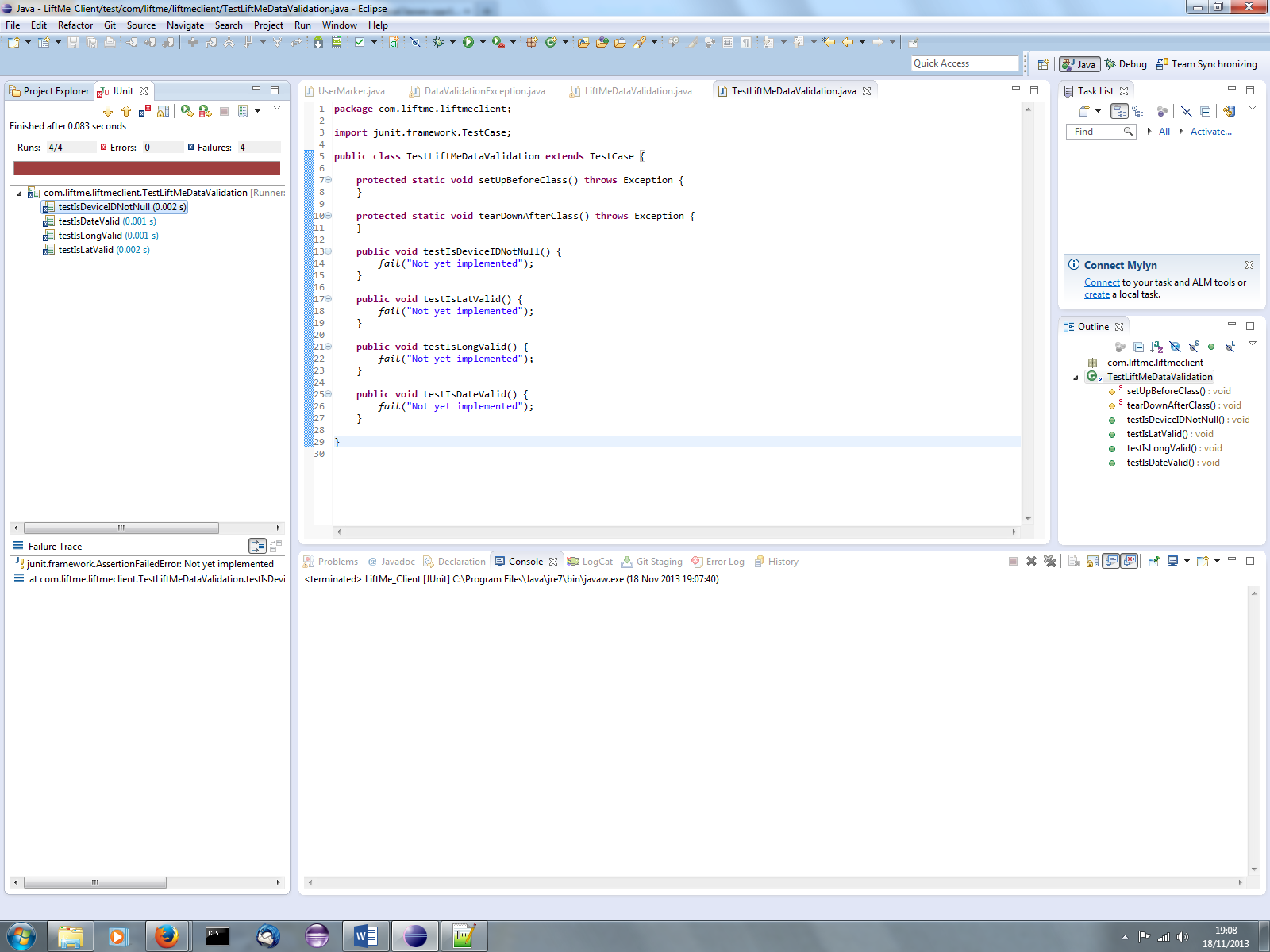
* testIsDeviceIDNotNull
* testIsLatValid
* testIsLongValid
* testIsDateValid

**TestLiftMeDataValidation Class Method Stubs**

|  |
| --- |
| **package** com.liftme.liftmeclient;  **import** junit.framework.TestCase;  **public** **class** TestLiftMeDataValidation **extends** TestCase {  **protected** **static** **void** setUpBeforeClass() **throws** Exception {  }  **protected** **static** **void** tearDownAfterClass() **throws** Exception {  }  **public** **void** testIsDeviceIDNotNull() {  *fail*("Not yet implemented");  }  **public** **void** testIsLatValid() {  *fail*("Not yet implemented");  }  **public** **void** testIsLongValid() {  *fail*("Not yet implemented");  }  **public** **void** testIsDateValid() {  *fail*("Not yet implemented");  }  } |

**Run JUnit Test Stubs**

We expect all of the tests to fail



Note that they have all failed (as expected).

**LiftMeDataValidation Class Method Stubs**

|  |
| --- |
| //171113 - MtpA - Created class  **package** com.liftme.liftmeclient;  **import** java.text.DateFormat;  **import** java.text.SimpleDateFormat;  **import** java.util.Date;  **public** **class** AppDataValidation {  **private** **static** DateFormat *dateFormat* = **new** SimpleDateFormat("yyyy-MM-dd HH:mm:ss");  **public** AppDataValidation() {    } // constructor - no args    **public** **static** **boolean** isDeviceIDNotNull(String vDeviceID) {  **return** **false**;  } // method isDeviceIDNotNull    **public** **static** **boolean** isLatValid(**double** vLatitude) {  **return** **false**;  } // method isLatValid    **public** **static** **boolean** isLongValid(**double** vLongditude) {  **return** **false**;  } // method isLongValid  **public** **static** **boolean** isDateValid(String vDate, String vTime) {  **return** **false**;  } // method checkValidDate  } // class AppDataValidation |

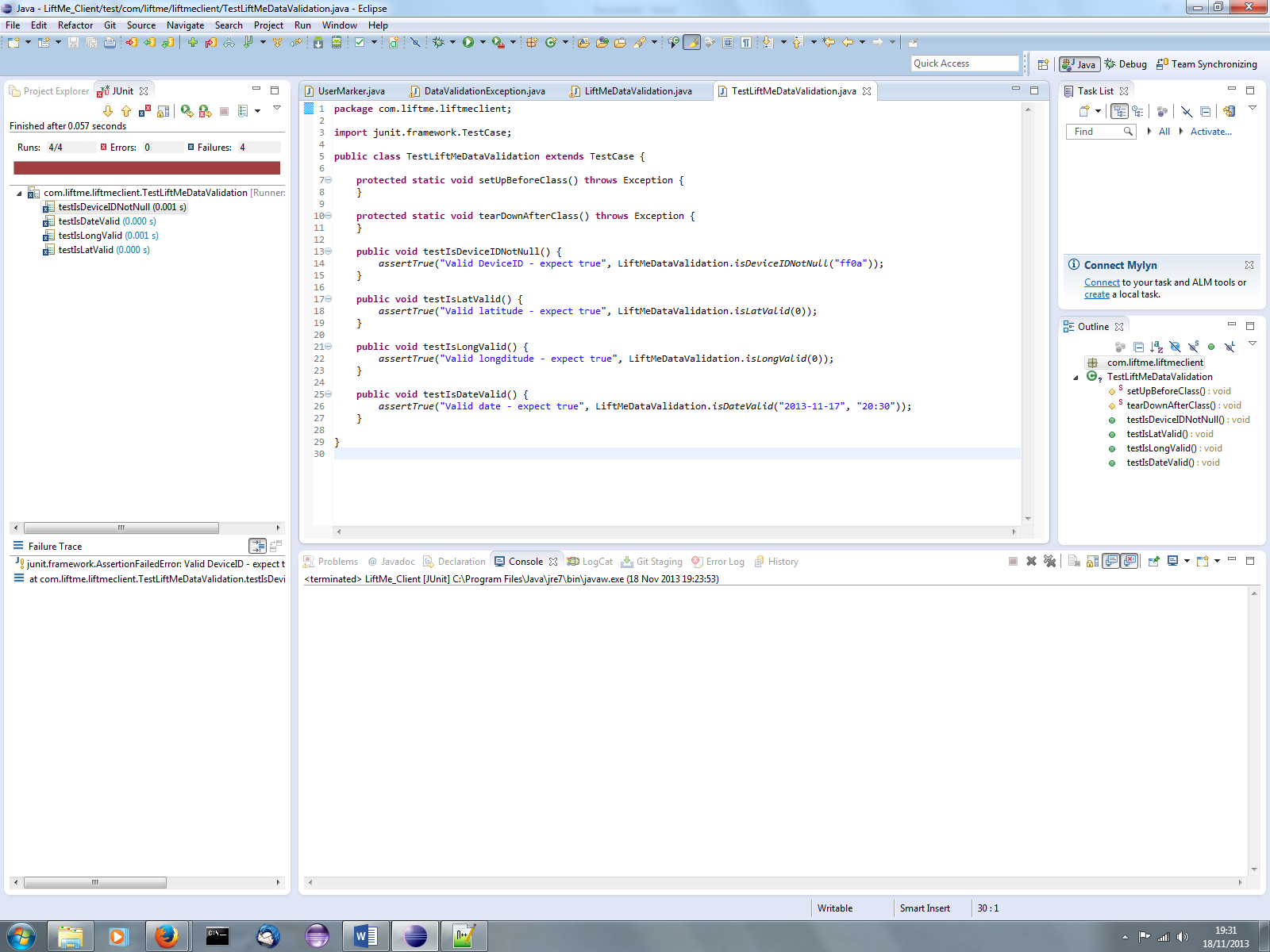
**TestLiftMeDataValidation Class Method Valid Tests**

The JUnit tests are all re-written to send valid data and expect ‘true’, but as the validation methods are just stubs and will always return false they should all fail.

|  |
| --- |
| **package** com.liftme.liftmeclient;  **import** junit.framework.TestCase;  **public** **class** TestLiftMeDataValidation **extends** TestCase {    **protected** **static** **void** setUpBeforeClass() **throws** Exception {  }  **protected** **static** **void** tearDownAfterClass() **throws** Exception {  }  **public** **void** testIsDeviceIDNotNull() {  *assertTrue*("Valid DeviceID - expect true", LiftMeDataValidation.*isDeviceIDNotNull*("ff0a"));  }  **public** **void** testIsLatValid() {  *assertTrue*("Valid latitude - expect true", LiftMeDataValidation.*isLatValid*(0));  }  **public** **void** testIsLongValid() {  *assertTrue*("Valid longditude - expect true", LiftMeDataValidation.*isLongValid*(0));  }  **public** **void** testIsDateValid() {  *assertTrue*("Valid date - expect true", LiftMeDataValidation.*isDateValid*("2013-11-17", "20:30"));  }  } |

**Run JUnit Test Stubs**

We expect them all to fail



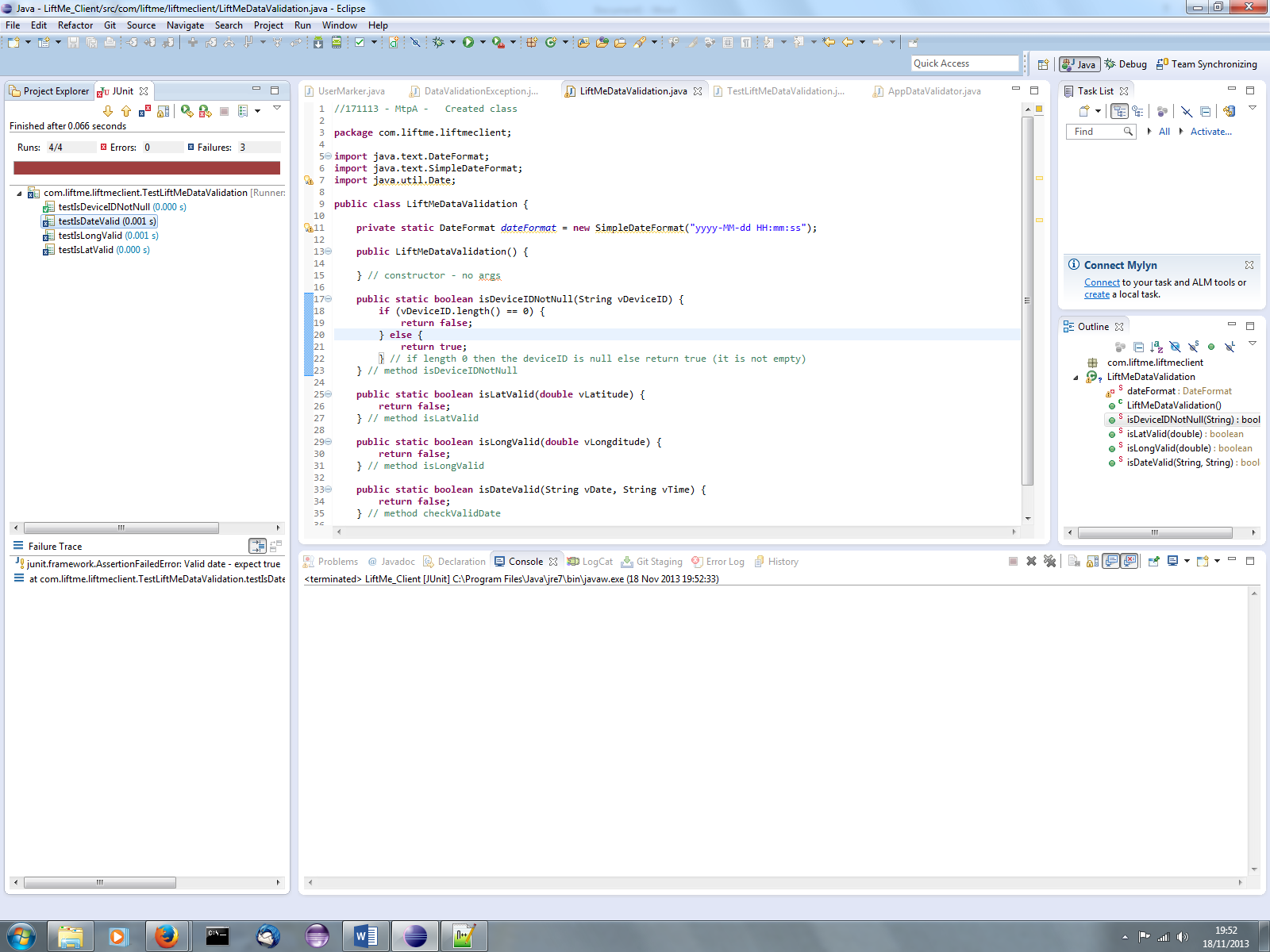
**Update LiftMeDataValidation – isDeviceIDNotNull method**

We now update this method and put in the code which we believe will pass our test.

|  |
| --- |
| //171113 - MtpA - Created class  **package** com.liftme.liftmeclient;  **import** java.text.DateFormat;  **import** java.text.SimpleDateFormat;  **import** java.util.Date;  **public** **class** LiftMeDataValidation {  **private** **static** DateFormat *dateFormat* = **new** SimpleDateFormat("yyyy-MM-dd HH:mm:ss");  **public** LiftMeDataValidation() {    } // constructor - no args    **public** **static** **boolean** isDeviceIDNotNull(String vDeviceID) {  **if** (vDeviceID.length() == 0) {  **return** **false**;  } **else** {  **return** **true**;  } // if length 0 then the deviceID is null else return true (it is not empty)  } // method isDeviceIDNotNull    **public** **static** **boolean** isLatValid(**double** vLatitude) {  **return** **false**;  } // method isLatValid    **public** **static** **boolean** isLongValid(**double** vLongditude) {  **return** **false**;  } // method isLongValid  **public** **static** **boolean** isDateValid(String vDate, String vTime) {  **return** **false**;  } // method checkValidDate  } // class LiftMeDataValidation |

**Re-run the JUnit Test Methods**

This time we are expecting the test on isDeviceIDNotNull to ***pass***

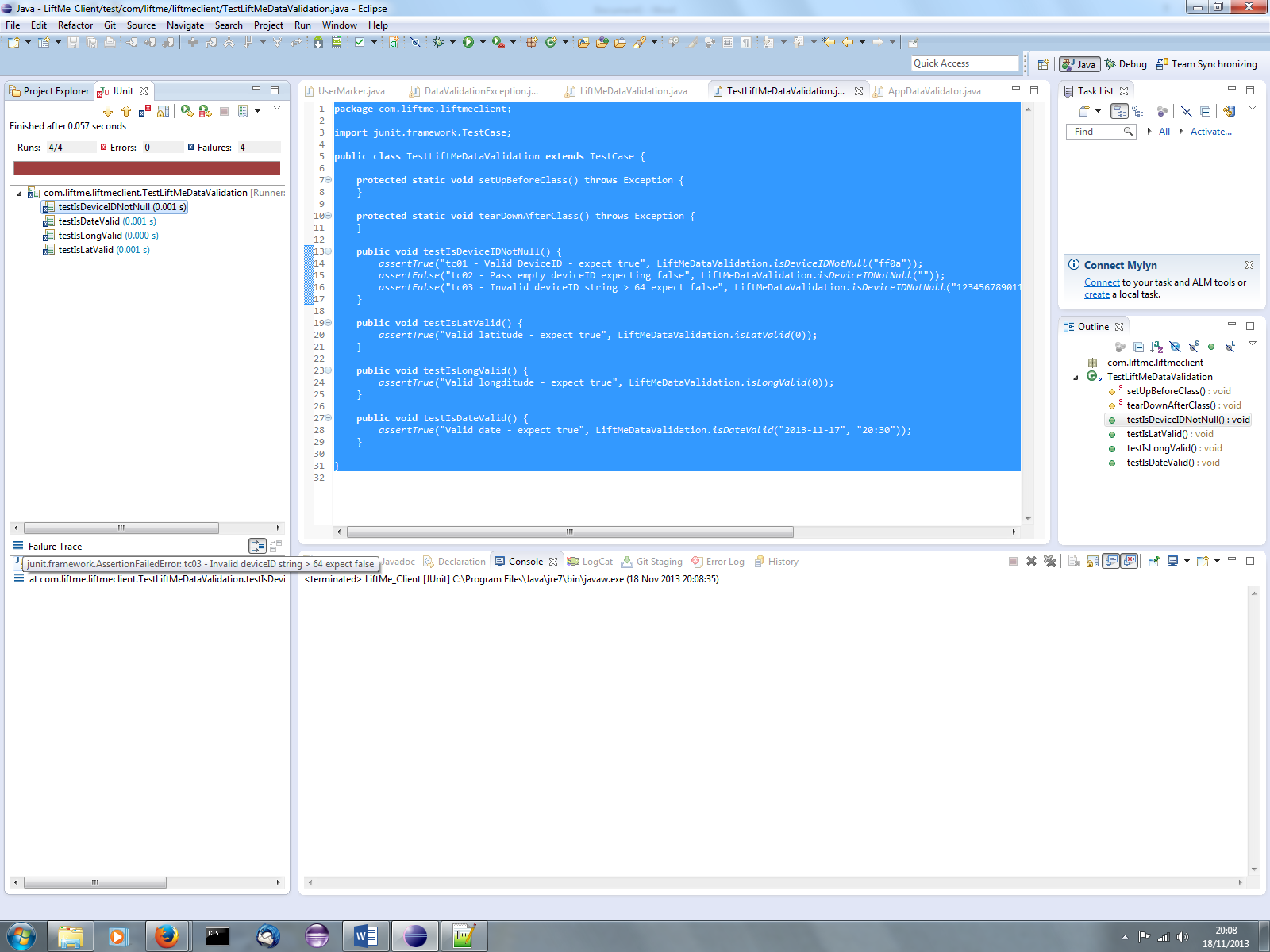


**Update the test method isDeviceIDNotNull to widen the test criteria**

* AssertFalse – when passing over a null string
* AssertFalse – when passing over a deviceID > 64 characters

|  |
| --- |
| **package** com.liftme.liftmeclient;  **import** junit.framework.TestCase;  **public** **class** TestLiftMeDataValidation **extends** TestCase {    **protected** **static** **void** setUpBeforeClass() **throws** Exception {  }  **protected** **static** **void** tearDownAfterClass() **throws** Exception {  }  **public** **void** testIsDeviceIDNotNull() {  *assertTrue*("tc01 - Valid DeviceID - expect true", LiftMeDataValidation.*isDeviceIDNotNull*("ff0a"));  *assertFalse*("tc02 - Pass empty deviceID expecting false", LiftMeDataValidation.*isDeviceIDNotNull*(""));  *assertFalse*("tc03 - Invalid deviceID string > 64 expect false", LiftMeDataValidation.*isDeviceIDNotNull*("123456789011234567890112345678901123456789011234567890112345678901"));  }  **public** **void** testIsLatValid() {  *assertTrue*("Valid latitude - expect true", LiftMeDataValidation.*isLatValid*(0));  }  **public** **void** testIsLongValid() {  *assertTrue*("Valid longditude - expect true", LiftMeDataValidation.*isLongValid*(0));  }  **public** **void** testIsDateValid() {  *assertTrue*("Valid date - expect true", LiftMeDataValidation.*isDateValid*("2013-11-17", "20:30"));  }  } |

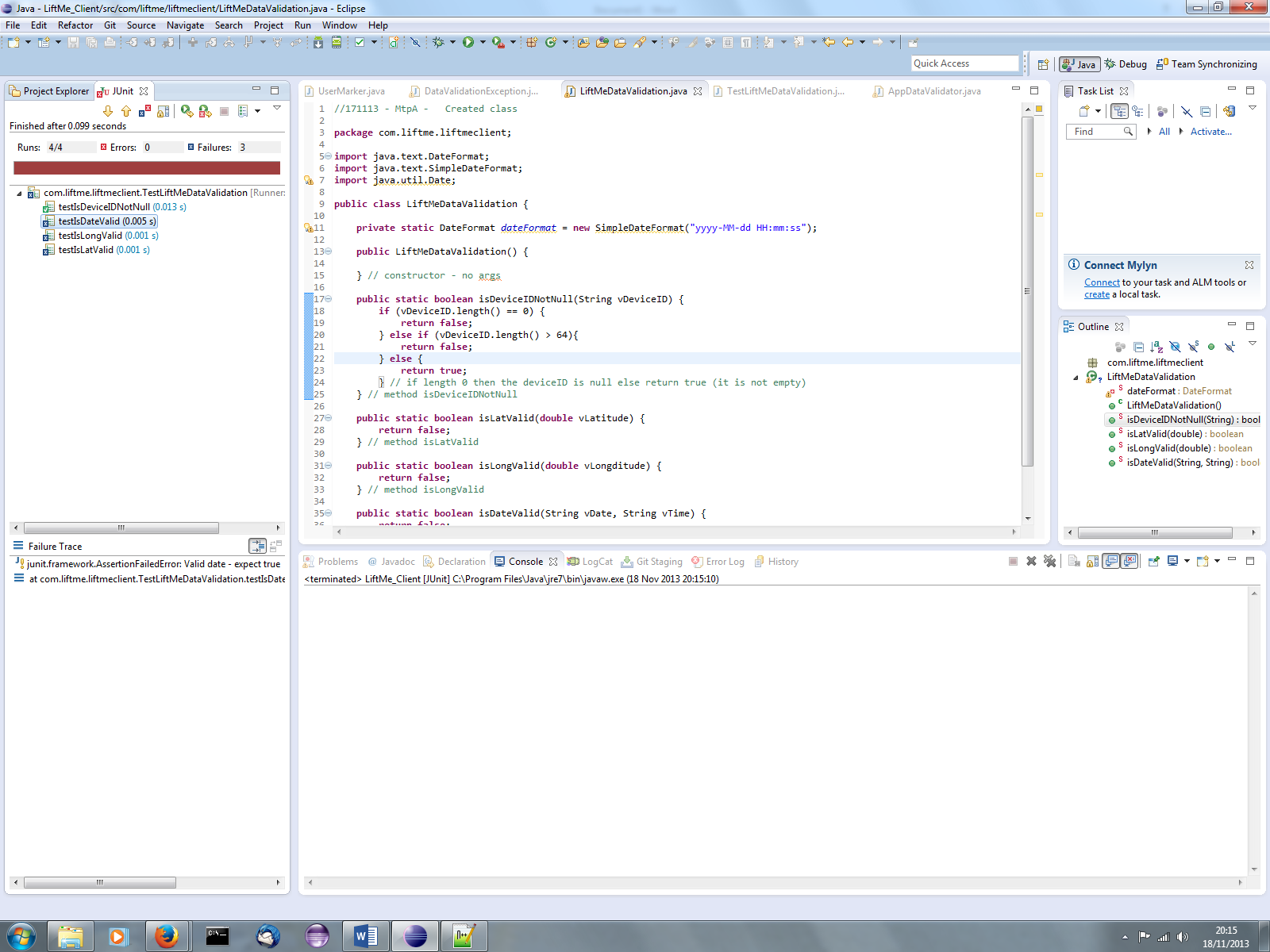
Expecting the test to pass. ***But it doesn’t ! It fails on the line > 64 characters, the value has been accepted.***



Therefore we now update the isDeviceIDNotNull method to include the condition to reject any deviceID which is longer than 64 characters.

|  |
| --- |
| //171113 - MtpA - Created class  **package** com.liftme.liftmeclient;  **import** java.text.DateFormat;  **import** java.text.SimpleDateFormat;  **import** java.util.Date;  **public** **class** LiftMeDataValidation {  **private** **static** DateFormat *dateFormat* = **new** SimpleDateFormat("yyyy-MM-dd HH:mm:ss");  **public** LiftMeDataValidation() {    } // constructor - no args    **public** **static** **boolean** isDeviceIDNotNull(String vDeviceID) {  **if** (vDeviceID.length() == 0) {  **return** **false**;  } **else** **if** (vDeviceID.length() > 64){  **return** **false**;  } **else** {  **return** **true**;  } // if length 0 or > 64 then the deviceID is invalid else return true (it is not empty)  } // method isDeviceIDNotNull    **public** **static** **boolean** isLatValid(**double** vLatitude) {  **return** **false**;  } // method isLatValid    **public** **static** **boolean** isLongValid(**double** vLongditude) {  **return** **false**;  } // method isLongValid  **public** **static** **boolean** isDateValid(String vDate, String vTime) {  **return** **false**;  } // method checkValidDate  } // class LiftMeDataValidation |

When we re-run with the amended code we can see that it now passes the testIsDeviceIDNotNull test

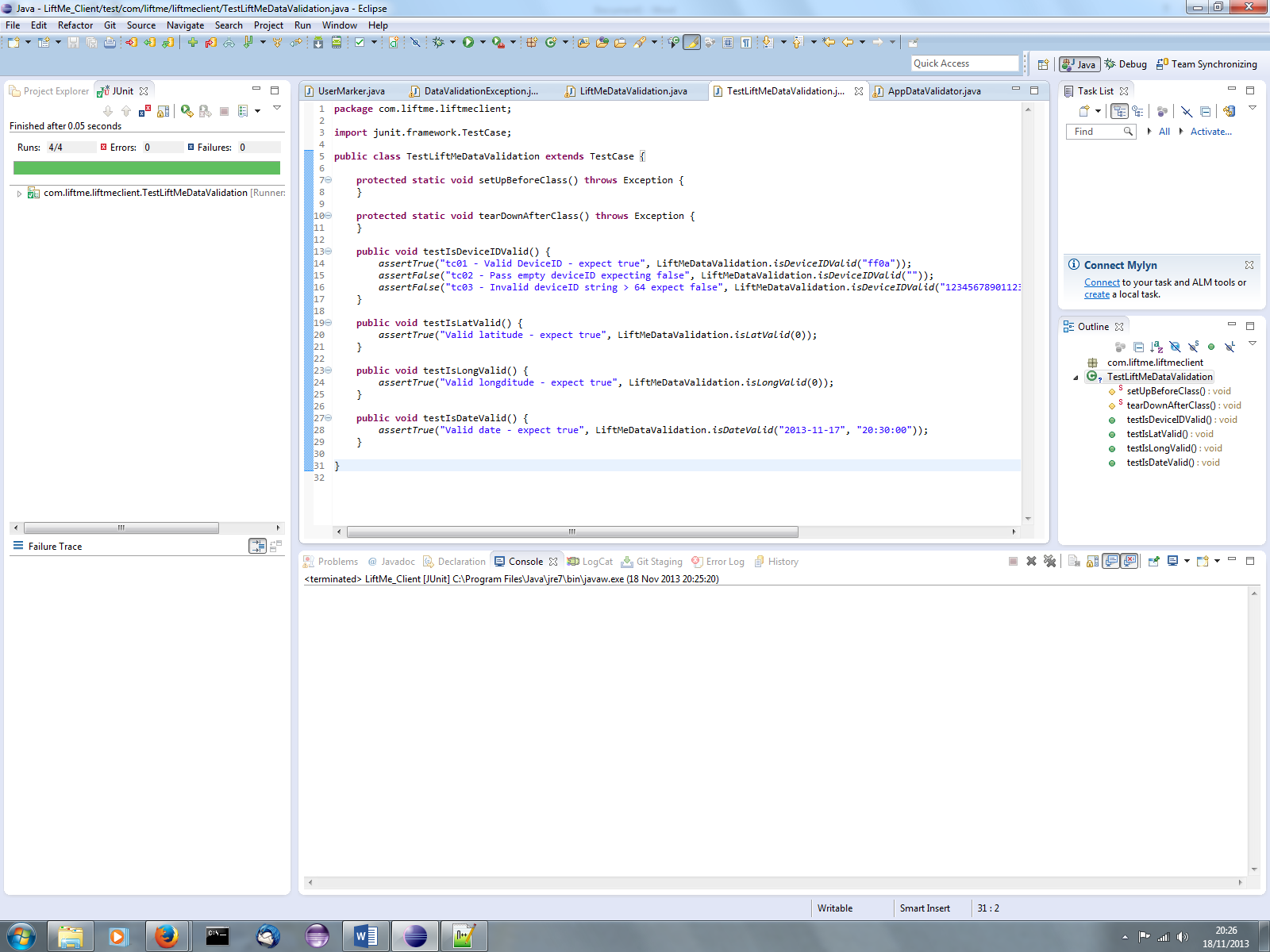


***Now*** we have to refactor the code to make the method name more meaningful, and rename the corresponding JUnit test.

|  |
| --- |
| **public** **static** **boolean** isDeviceIDValid(String vDeviceID) {  **if** (vDeviceID.length() == 0) {  **return** **false**;  } **else** **if** (vDeviceID.length() > 64){  **return** **false**;  } **else** {  **return** **true**;  } // if length 0 or > 64 then the deviceID is invalid else return true (it is not empty)  } // method isDeviceIDValid  **public** **void** testIsDeviceIDValid() {  *assertTrue*("tc01 - Valid DeviceID - expect true", LiftMeDataValidation.*isDeviceIDValid*("ff0a"));  *assertFalse*("tc02 - Pass empty deviceID expecting false", LiftMeDataValidation.*isDeviceIDValid*(""));  *assertFalse*("tc03 - Invalid deviceID string > 64 expect false", LiftMeDataValidation.*isDeviceIDValid*("123456789011234567890112345678901123456789011234567890112345678901"));  } |

Repeat the process with the remaining methods until all of the tests have passed

|  |
| --- |
| //171113 - MtpA - Created class  **package** com.liftme.liftmeclient;  **import** java.text.DateFormat;  **import** java.text.SimpleDateFormat;  **import** java.util.Date;  **public** **class** LiftMeDataValidation {  **private** **static** DateFormat *dateFormat* = **new** SimpleDateFormat("yyyy-MM-dd HH:mm:ss");  **public** LiftMeDataValidation() {    } // constructor - no args    **public** **static** **boolean** isDeviceIDValid(String vDeviceID) {  **if** (vDeviceID.length() == 0) {  **return** **false**;  } **else** **if** (vDeviceID.length() > 64){  **return** **false**;  } **else** {  **return** **true**;  } // if length 0 or > 64 then the deviceID is invalid else return true (it is not empty)  } // method isDeviceIDValid    **public** **static** **boolean** isLatValid(**double** vLatitude) {  **if** (vLatitude < -90.0 || vLatitude > 90.0) {  **return** **false**;  } **else** {  **return** **true**;  } // if lt -90 or gt 90 is not valid  } // method isLatValid    **public** **static** **boolean** isLongValid(**double** vLongditude) {  **if** (vLongditude < -180.0 || vLongditude > 180.0) {  **return** **false**;  } **else** {  **return** **true**;  } // if lt -180 or gt 180 is not valid  } // method isLongValid  **public** **static** **boolean** isDateValid(String vDate, String vTime) {  *dateFormat*.setLenient(**false**);  **try** {  Date validDate = *dateFormat*.parse(vDate + " " + vTime);  **return** **true**;  } **catch** (Exception exDate) {  **return** **false**;  } // try to convert to a date, if can then OK otherwise it is invalid  } // method checkValidDate  } // class LiftMeDataValidation |



**Additional test cases :**

The illustrative walkthrough above was to indicate the process and to plug in real data taken from the project. However, the actual test unit contains the following code for more extensive testing across the data ranges. (Note : Even this is not fully comprehensive as not all date YYYY-MM-DD HH:MM:SS combinations have been tried).

|  |
| --- |
| **package** com.liftme.liftmeclient;  **import** junit.framework.TestCase;  **public** **class** TestLiftMeDataValidation **extends** TestCase {    **protected** **static** **void** setUpBeforeClass() **throws** Exception {  }  **protected** **static** **void** tearDownAfterClass() **throws** Exception {  }  **public** **void** testIsDeviceIDValid() {  *assertTrue*("tc01 - Valid DeviceID - expect true", LiftMeDataValidation.*isDeviceIDValid*("ff0a"));  *assertFalse*("tc02 - Pass empty deviceID expecting false", LiftMeDataValidation.*isDeviceIDValid*(""));  *assertFalse*("tc03 - Invalid deviceID string > 64 expect false", LiftMeDataValidation.*isDeviceIDValid*("123456789011234567890112345678901123456789011234567890112345678901"));  }  **public** **void** testIsLatValid() {  *assertTrue*("tc04 - Valid latitude - expect true", LiftMeDataValidation.*isLatValid*(0));  *assertTrue*("tc05 - Valid latitude - expect true", LiftMeDataValidation.*isLatValid*(-90));  *assertFalse*("tc06 - Invalid latitude - expect false", LiftMeDataValidation.*isLatValid*(-90.1));  *assertTrue*("tc07 - Valid latitude - expect true", LiftMeDataValidation.*isLatValid*(-10.0567));  *assertTrue*("tc08 - Valid latitude - expect true", LiftMeDataValidation.*isLatValid*(2.00000123));  *assertTrue*("tc09 - Valid latitude - expect true", LiftMeDataValidation.*isLatValid*(89.99999));  *assertFalse*("tc10 - Invalid latitude - expect false", LiftMeDataValidation.*isLatValid*(90000.1234));  }  **public** **void** testIsLongValid() {  *assertTrue*("tc11 - Valid longditude - expect true", LiftMeDataValidation.*isLongValid*(0));  *assertTrue*("tc12 - Valid longditude - expect true", LiftMeDataValidation.*isLongValid*(-180));  *assertFalse*("tc13 - Invalid longditude - expect false", LiftMeDataValidation.*isLongValid*(-180.1));  *assertTrue*("tc14 - Valid longditude - expect true", LiftMeDataValidation.*isLongValid*(-10.0567));  *assertTrue*("tc15 - Valid longditude - expect true", LiftMeDataValidation.*isLongValid*(2.00000123));  *assertTrue*("tc16 - Valid longditude - expect true", LiftMeDataValidation.*isLongValid*(179.99999));  *assertFalse*("tc17 - Invalid longditude - expect false", LiftMeDataValidation.*isLongValid*(90000.1234));  }  **public** **void** testIsDateValid() {  *assertTrue*("tc18 - Valid date - expect true", LiftMeDataValidation.*isDateValid*("2013-11-17", "20:30:00"));  *assertTrue*("tc19 - Valid date - expect true", LiftMeDataValidation.*isDateValid*("1901-11-17", "20:30:00"));  *assertTrue*("tc20 - Valid date - expect true", LiftMeDataValidation.*isDateValid*("2013-01-01", "00:00:00"));  *assertFalse*("tc21 - Invalid date - expect false", LiftMeDataValidation.*isDateValid*("a", "b"));  *assertFalse*("tc22 - Invalid date - expect false", LiftMeDataValidation.*isDateValid*("2013-02-30", "19:00:00"));  *assertFalse*("tc23 - Invalid date - expect false", LiftMeDataValidation.*isDateValid*("2050-02-02", "19:00:00"));  *assertTrue*("tc24 - Valid date - expect true", LiftMeDataValidation.*isDateValid*("2012-02-29", "10:00:00"));  *assertFalse*("tc25 - Invalid date - expect false", LiftMeDataValidation.*isDateValid*("2013-00-01", "19:00:00"));  *assertFalse*("tc26 - Invalid date - expect false", LiftMeDataValidation.*isDateValid*("2013-13-01", "19:00:00"));  *assertFalse*("tc27 - Invalid date - expect false", LiftMeDataValidation.*isDateValid*("2013-01-32", "19:00:00"));  *assertFalse*("tc28 - Invalid date - expect false", LiftMeDataValidation.*isDateValid*("2013-04-31", "19:00:00"));  *assertFalse*("tc29 - Invalid date - expect false", LiftMeDataValidation.*isDateValid*("2013-01-00", "19:00:00"));  }  } |

**Next stage of testing – Component :**

The next step is to update the Class UserMarker to validate the constructor when it is built with data. The same steps on how to build up the test cases are followed but an example end point is given below. (For the document purpose there are just a couple of tests to illustrate the principal).

The three coding changes are required :

* DataValidationException Class

Create a custom exception – easier to track the specific errors

|  |
| --- |
| **package** com.liftme.liftmeclient;  **public** **class** DataValidationException **extends** Exception {  **public** DataValidationException() {    } // constructor - no args    } |

* UserMarker Class – Add the validation checks to the build() method

Add calls to the following methods

* + LiftMeDataValidator.isDeviceIDValid
  + LiftMeDataValidator.isLatValid
  + LiftMeDataValidator.isLongValid
  + LiftMeDataValidator.isDateValid

|  |
| --- |
| **public** UserMarker build() **throws** DataValidationException {  **if** (checkValidFields()) {  **return** **new** UserMarker(**this**);  } **else** {  **throw** **new** DataValidationException();  }  } // method build    **private** **boolean** checkValidFields() {  **if** (!LiftMeDataValidator.*isDeviceIDValid*(**this**.deviceID)) {  **return** **false**;  }  **if** (!LiftMeDataValidator.*isLatValid*(**this**.markerLat)) {  **return** **false**;  }  **if** (!LiftMeDataValidator.*isLongValid*(**this**.markerLong)) {  **return** **false**;  }  **return** LiftMeDataValidator.*isDateValid*(**this**.markerDate, **this**.markerTime);  } // method checkValidFields |

* TestUserMarker – Junit test class

Create tests for the DataValidationException class being thrown when the UserMarker builder is invoked.

|  |
| --- |
| **package** com.liftme.liftmeclient;  **import** java.util.Calendar;  **import** org.junit.Test;  **import** junit.framework.TestCase;  **public** **class** TestUserMarker **extends** TestCase {  **protected** **static** **void** setUpBeforeClass() **throws** Exception {  }  **protected** **static** **void** tearDownAfterClass() **throws** Exception {  }  @Test  **public** **void** testUserMarkerBuilder() {  Calendar currDate = Calendar.*getInstance*();  **try** {  UserMarker testDevice = **new** UserMarker.Builder("fa001", 0, 0, currDate).build();  } **catch** (DataValidationException buildEx) {  *fail*("Device ID fail in UserMarker constructor");  }  }    @Test  **public** **void** testUserMarkerBuilderDeviceIDFail() {  Calendar currDate = Calendar.*getInstance*();  **try** {  UserMarker testDevice = **new** UserMarker.Builder("", 0, 0, currDate).build();  *fail*("Should have thrown exception");  } **catch** (DataValidationException e) {  ;  }  }    } |

Note :

* Method testUserMarkerBuilder()

Fails if an exception is thrown. I.e. The test is designed to pass

* Method testUserMarkerBuilderDeviceIDFail()

Fails if ***no*** exception is thrown. I.e. The test is designed to ***fail !***