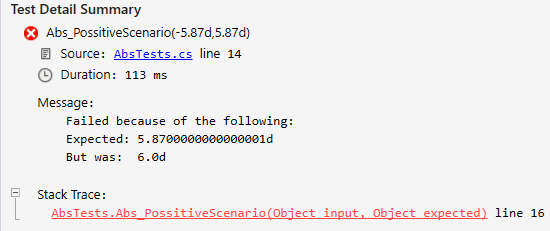
**BUG REPORT OF UNIT TESTING**

**Bug\_1:** ABS function of the calculator returns rounded values for the fractional numbers.

Calculator1.Abs(-5.87) returns 6.0 value.

**Steps**:

1. Run Test “Abs\_PossitiveScenario” for Calculator1.dll with fractional number (e.g. -5.87)

**Actual Result:** Assert.AreEqual fails  


**Expected Result**: Calculator1.Abs(-5.87) returns 5.87, Assert.AreEqual passes for the test “Abs\_PossitiveScenario”.

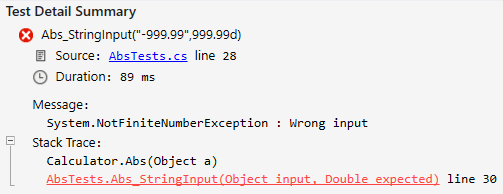
----------------------------------------------------------------------------------------------------------------------------------------------

**Bug\_2:** ABS function does not accept frunctional numbers with string type of input.

Frunctional numbers with type of string input are not accepted for the Calculator1.Abs (e.g. “-999.99”), there is no problem with an integer number (e.g. “-999”).

**Steps**:

1. Run Test “Abs\_StringInput” for Calculator1.dll with fractional number in string type of variable (e.g: “-999.99”)

**Actual Result:** Calculator1.Abs fails because of the wront input (system error):  


**Expected Result**: Calculator1.Abs(“-999.99”) returns 999.99, Assert.AreEqual passes for the test “Abs\_StringInput”.

----------------------------------------------------------------------------------------------------------------------------------------------

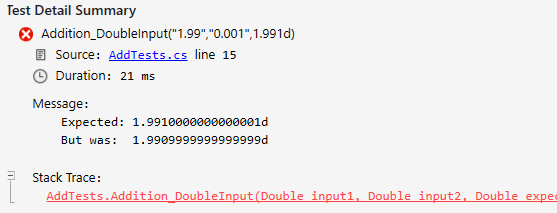
**Bug\_3:** There is a rounding issue with Frunctional number 0.001 in Calculator1.Add.

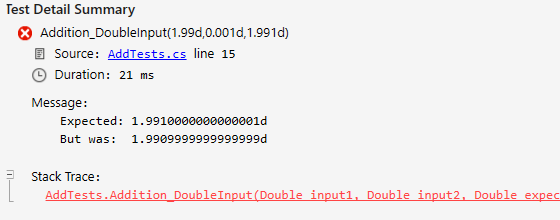
Addition of frunctional numbers fails for the Calculator1.Add (e.g. 1.99 + 0.001), there is no problem with the fractional number 0.002 (e.g. 1.99 + 0.002 = 1.992).

**Steps**:

1. Run Test “Add\_DoubleInput” for Calculator1.dll with fractional numbers 1.99 and 0.001

**Actual Result:** Assert.AreEqual fails because of the rounding issue:

String input:  


Double input:  


**Expected Result**: Calculator1.Add(1.99, 0.001) returns 1.991, Assert.AreEqual passes for the test “Add\_DoubleInput”.

----------------------------------------------------------------------------------------------------------------------------------------------

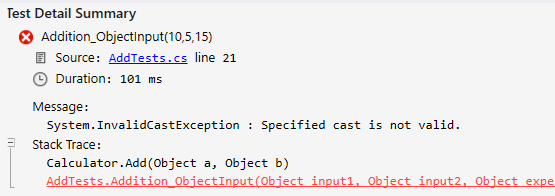
**Bug\_4:** Object/Int type of input is not accepted for Calculator1.Add.

There is an issue with entering object, int type of values for Calculator1.Add. System.InvalidCastException : Specified cast is not valid.

**Steps**:

1. Run Test “Add\_ObjectInput” for Calculator1.dll with integer numbers with object or integer type.

**Actual Result:** Calculator.Add fails because of the system exception:



**Expected Result**: Calculator1.Add accepts any value with object or integer type, Assert.AreEqual passes for the test “Add\_ObjectInput”.

----------------------------------------------------------------------------------------------------------------------------------------------

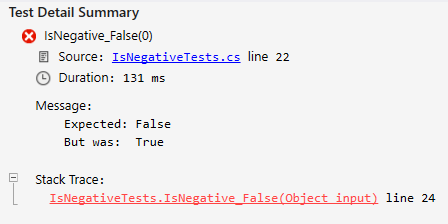
**Bug\_5:** Calculator1.isNegative return True for Zero

Zero is not positive or negative but Calculator1.isNegative recognizes it as negative value.

**Steps**:

1. Run Test “IsNegative\_False” for Calculator1.dll with Input = 0.

**Actual Result:** Assert.IsFalse returns True:



**Expected Result**: Calculator1.isNegative doesn’t recognize Zero as negative value, Assert.IsFalse passes for the test “IsNegative\_False”.

----------------------------------------------------------------------------------------------------------------------------------------------

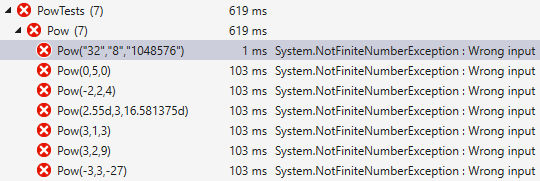
**Bug\_6:** Calculator1.Pow does not work with any input.

Calculator1.Pow does not work with any input (object, string, double).

**Steps**:

1. Run Test “Pow” for Calculator1.dll

**Actual Result:** Calculator1.Pow return system exception for any Input:



**Expected Result**: Calculator1.Pow calculates object, string, integer or double input, Assert.AreEqual passes for the test “Pow”.

----------------------------------------------------------------------------------------------------------------------------------------------

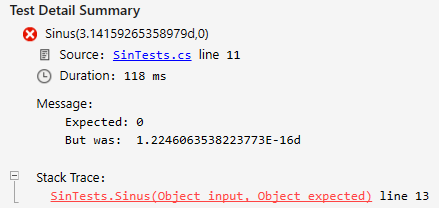
**Bug7:** Calculator1.Sin(PI) return wrong value.

Sin(pi) return 1.2246063538223773E-16d instead of zero.

**Steps**:

1. Run Test “Sinus” for Calculator1.dll with input = 3.14159265358979323846

**Actual Result:** Assert.AreEqual fails:



**Expected Result**: Calculator1.Sin(3.14159265358979323846) return zero, Assert.AreEqual passes for the test “Sinus”.