Unit-testing exercises

The focus of today is on testing. Where possible, you should solve the exercises below using testdriven development[1] and you should aim for 100% code coverage. If that is not attainable, you should be able to explain why.

1. Explain, in your own words, the concept of exceptions.
2. Explain, in your own words, the concept of code coverage.
3. Explain, in your own words, the process of test-driven development.

4. Write a class to represent a circle. A circle has a center and a radius. Add a method to determine if a point (x, y) is inside the circle. Add a method to determine if two circles overlap.

5. Write tests and correct the bad code in BadList.cs. Find BadList.cs here: https://git.io/JTxO6

6. Write a class to represent a 2d vector. Add an appropriate constructor and methods for addition and subtraction of vectors. Add methods to compute the scalar and cross product.

7. Write your own String class. Internally the class should use an array of characters to represent the string. Add the following methods: CharAt, length, substring, ToLowerCase, and Equals. You must not use any part of the .NET string type in your implementation.

8. Write a class to represent a time duration. Internally the class should store the time in miliseconds. The class should expose two constructors:
   - Duration(long miliseconds),
   - Duration(long hours, long minutes, long seconds), and

   Add properties to the class which return the different **components** of the duration:
   - long MiliSeconds
   - long Seconds
   - ...
   - long Years

   Add a total-variant of the above properties, eg. TotalSeconds

   Add methods to the class which allow manipulation of time units.

   - Duration Add(Duration d) which adds two durations and returns the result.
   - Duration Sub(Duration d) which subtracts two durations and returns the result.
   - Duration Mul(int m) which multiplies the duration by a factor of m.

9. Write a class to represent a stack of integers. Internally the class should use an array of integers to represent the stack. Add methods for pushing and popping elements of the stack.

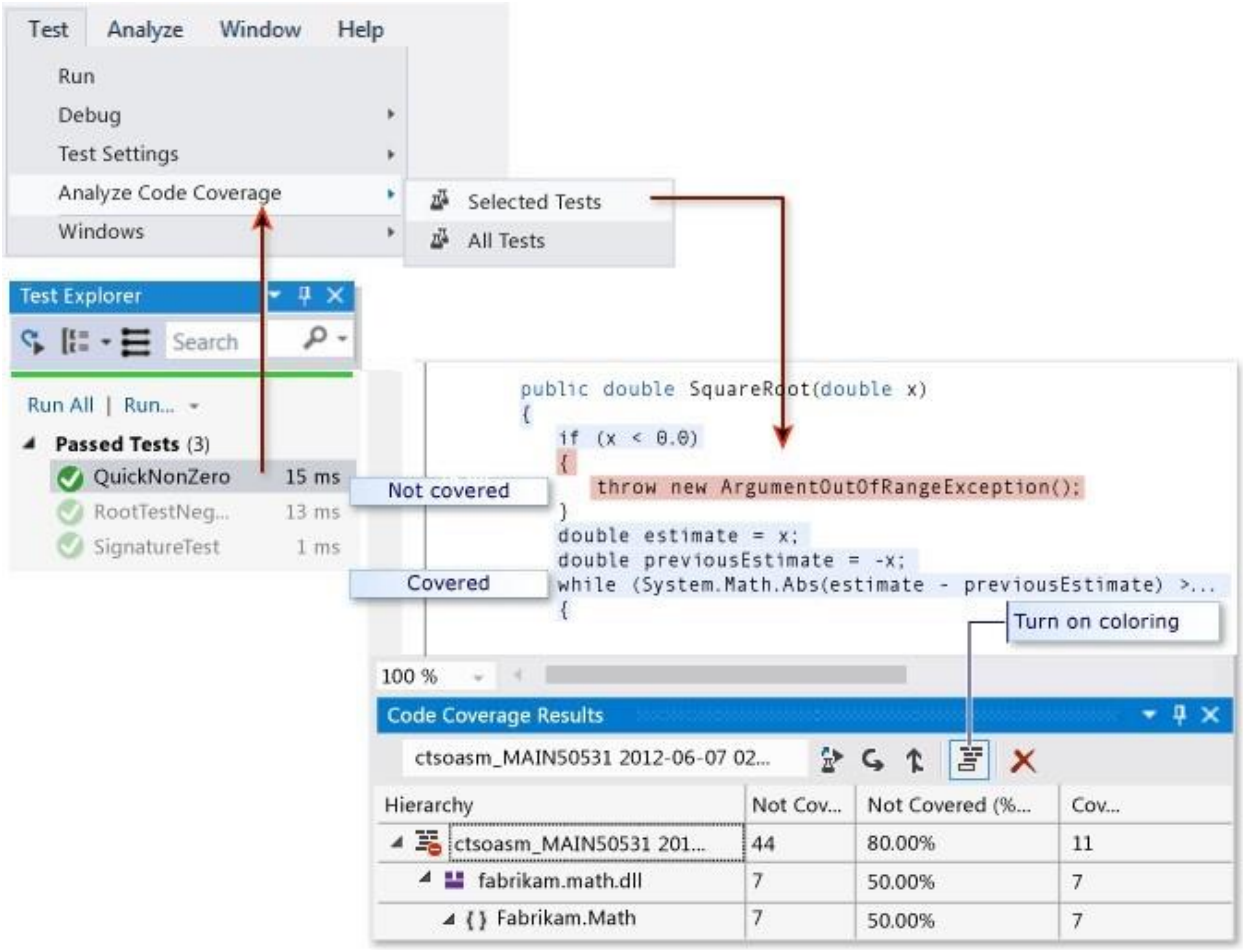
---

1 See slides or https://en.wikipedia.org/wiki/Test-driven_development

See: https://en.wikipedia.org/wiki/Stack_(abstract_data_type)

10. Rewrite the class from the previous problem to internally store the time in hours, minutes, seconds, and nanoseconds. This is a painful refactoring, but thanks to your many test cases, you should feel confident that your changes are correct.

11. Write tests for last week's exercises – a test-project for some of the exercises already exists


Test
Analyze
Window
Help
Run
Debug
Test Settings
Analyze Code Coverage
Windows
Selected Tests
All Tests
Test Explorer
Search
Run All | Run...
Passed Tests (3)
QuickNonZero 15 ms
RootTestNeg... 13 ms
SignatureTest 1 ms
Not covered
Covered
public double SquareRoot(double x)
{
if (x < 0.0)
{
throw new ArgumentOutOfRangeException();
}
double estimate = x;
double previousEstimate = -x;
while (System.Math.Abs(estimate - previousEstimate) >...
{
Turn on coloring
100 %
Code Coverage Results
ctsoasm_MAIN50531 2012-06-07 02...
Hierarchy
Not Cov...
Not Covered (%...
Cov...
ctsoasm_MAIN50531 201...
44
80.00%
11
fabrikam.math.dll
7
50.00%
7
Fabrikam.Math
7
50.00%
7