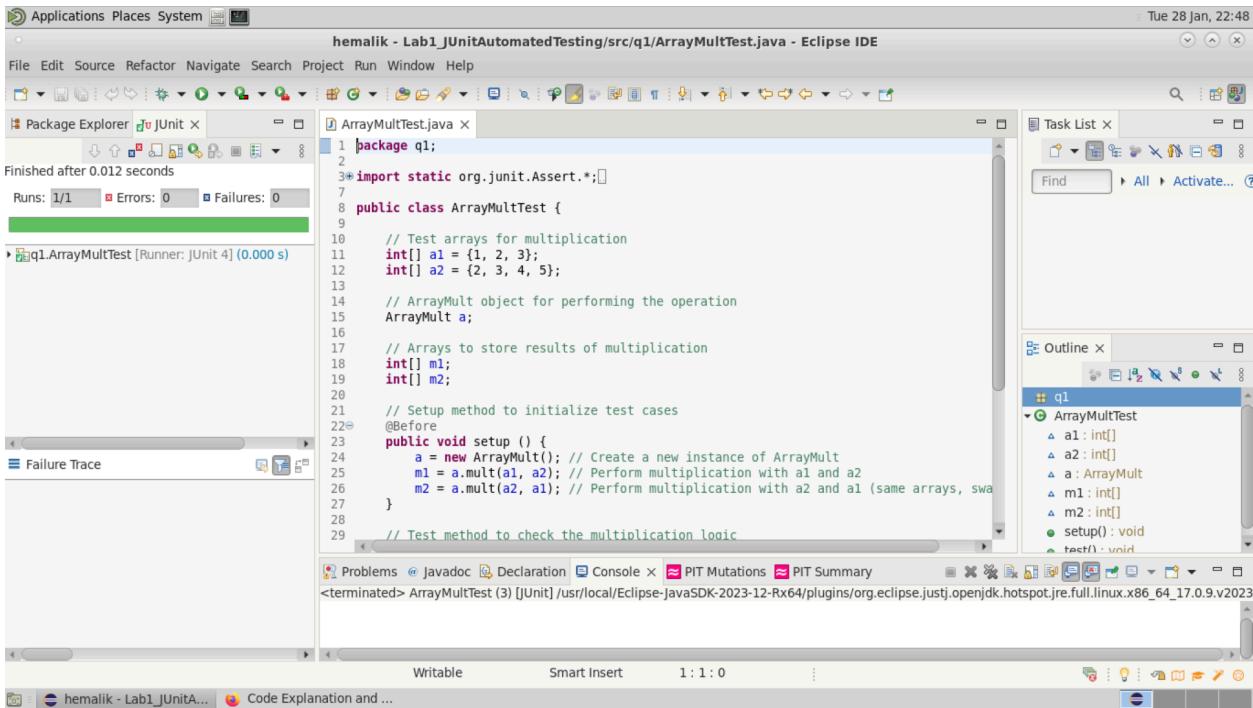


COE891 Lab I: JUnit: Automated Testing
Hamza Malik
501112545
Jan 28, 2025



```
hemalik - Lab1_JUnitAutomatedTesting/src/q1/ArrayMultTest.java - Eclipse IDE
Tue 28 Jan, 22:48

File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer JUnit X
Finished after 0.012 seconds
Runs: 1/1 Errors: 0 Failures: 0
q1.ArrayMultTest [Runner: JUnit 4] (0.000 s)

1 package q1;
2
3 import static org.junit.Assert.*;
4
5 public class ArrayMultTest {
6
7     // Test arrays for multiplication
8     int[] a1 = {1, 2, 3};
9     int[] a2 = {2, 3, 4, 5};
10
11    // ArrayMult object for performing the operation
12    ArrayMult a;
13
14    // Arrays to store results of multiplication
15    int[] m1;
16    int[] m2;
17
18    // Setup method to initialize test cases
19    @Before
20    public void setup () {
21        a = new ArrayMult(); // Create a new instance of ArrayMult
22        m1 = a.mult(a1, a2); // Perform multiplication with a1 and a2
23        m2 = a.mult(a2, a1); // Perform multiplication with a2 and a1 (same arrays, swapped)
24    }
25
26    // Test method to check the multiplication logic
27
28
29 }

Problems Javadoc Declaration Console X PIT Mutations PIT Summary
<terminated> ArrayMultTest (3) [JUnit] /usr/local/Eclipse-JavaSDK-2023-12-Rx64/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.linux.x86_64_17.0.9.v2023

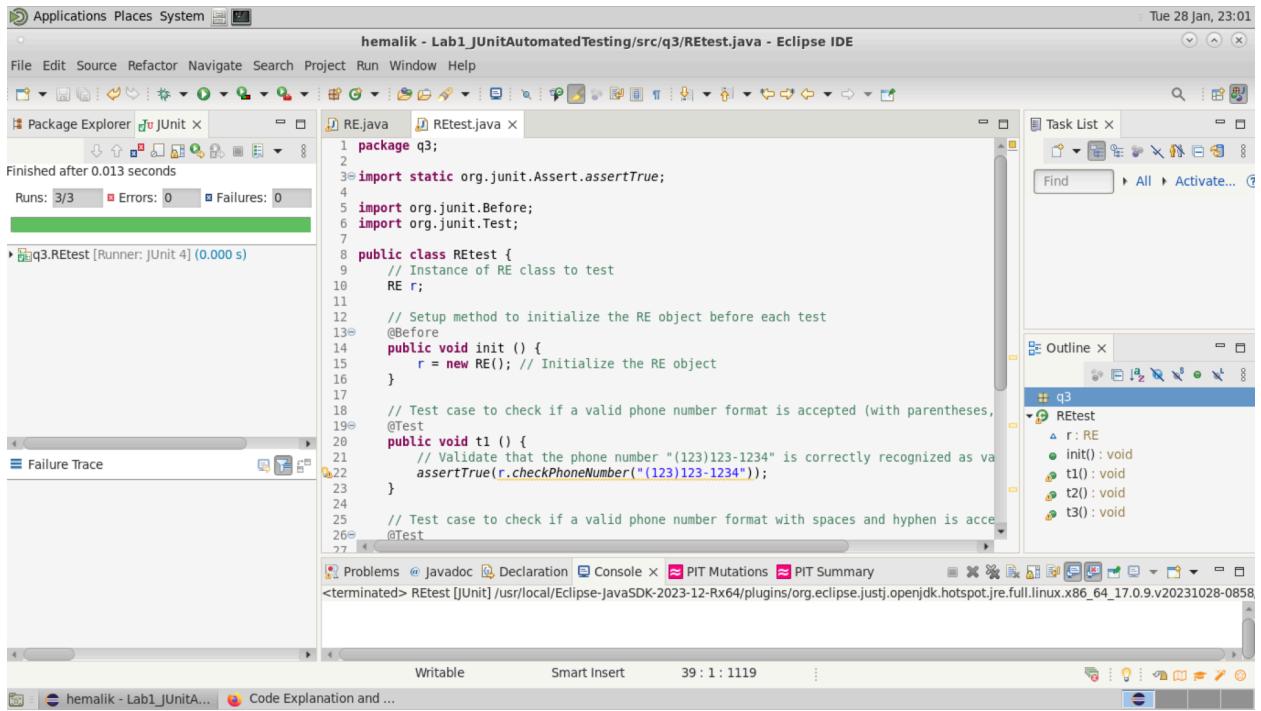
Writable Smart Insert 1:1:0
```

Q1: In this task, we implemented the `mult(int[] array1, int[] array2)` method, which performs element-wise multiplication of two integer arrays while accommodating different lengths. The resulting output array takes the length of the longer input array. Where both arrays have values at the same index, their corresponding elements are multiplied. If one array is shorter, any remaining elements from the longer array are directly copied into the output. A JUnit test class is used to verify the correctness of the `ArrayMult` implementation, ensuring it handles different input sizes and produces accurate results.

The screenshot shows the Eclipse IDE interface with the following details:

- Title Bar:** hemalik - Lab1_JUnitAutomatedTesting/src/q2/TriangleTest.java - Eclipse IDE
- Toolbar:** Standard Eclipse toolbar with icons for file operations, search, and project navigation.
- Left Sidebar:** Package Explorer showing the project structure. It lists "q2" and "TriangleTest.java". Below it, a message says "Finished after 0.023 seconds" and displays "Runs: 6/6 Errors: 0 Failures: 0".
- Central Area:** Editor view showing the code for `TriangleTest.java`. The code includes imports for `org.junit.Assert.*`, a class definition for `TriangleTest`, and a `@Before` annotated method `setup` which initializes several `Triangle` objects (t1 through t5) with various side lengths, including invalid ones.
- Right Sidebar:** Outline view showing the class structure: `q2` contains `TriangleTest`, which has fields `t1`, `t2`, `t3`, `t4`, and `t5`, along with methods `setup()` and `testTriValid()`.
- Bottom Status Bar:** Shows "Writable", "Smart Insert", and the current time "1:1:0".
- Bottom Navigation Bar:** Includes tabs for "Problems", "Javadoc", "Declaration", "Console", "PIT Mutations", and "PIT Summary".

Q2: In this task, we were given a `Triangle` class with a `calculateArea` method that incorrectly implemented Heron's formula. Our first objective was to identify and fix the error in the mathematical equation. Once corrected, we were required to use JUnit testing to validate the implementation. This included verifying whether a triangle with sides (3, 4, 100) is valid, writing three test methods to compute the area of triangles `t1`, `t2`, and `t3`, and checking for both positive and negative test cases. Additionally, we tested whether the area of `t1` was equal to `t2` to ensure consistency in our calculations. The JUnit tests provided comprehensive validation for the corrected `calculateArea` method and ensured its accuracy in handling various triangle inputs.



Q3: In this task, we implemented a class with a method to validate phone numbers using regular expressions. The `checkPhoneNumber` method uses the `matches` function to verify whether an input follows the expected pattern: three digits, a space, three digits, a hyphen, and four digits (e.g., `123 456-7890`). The program prompts the user to enter a phone number, then calls the `checkPhoneNumber` method to determine its validity and prints the result. The issue with the initial regular expression was that it did not account for potential variations in spacing around parentheses and the hyphen, leading to incorrect validation. The regex `(\d{3}) \d{3} - \d{4}` enforced a strict pattern with exactly one space between the area code and the first segment, as well as before the hyphen. To fix this, we needed to modify the regex to allow flexible spacing while ensuring the correct structure of the phone number.

The screenshot shows the Eclipse IDE interface with the following details:

- Title Bar:** hemalik - Lab1_JUnitAutomatedTesting/src/q4/suite.java - Eclipse IDE
- Toolbar:** Standard Eclipse toolbar with icons for file operations, search, and project management.
- Package Explorer:** Shows a single package named "q4".
- Suite.java Content:** The code defines a JUnit test suite named "suite". It imports org.junit.runner.RunWith and org.junit.runners.Suite, and then defines a @Suite.SuiteClasses block containing two test classes: q2.TriangleTest and q3.REtest. It also includes annotations for running multiple test classes and specifying the test classes to be included.
- Console View:** Displays the output of the test run:

```
<terminated> suite [JUnit] /usr/local/Eclipse-JavaSDK-2023-12-Rx64/plugins/org.eclipse.jst.jdt.openjdk.hotspot.jre.full.linux.x86_64_17.0.9.v20231028-0858/junit
s=9.0
result = 12.0
```
- Outline View:** Shows the structure of the "suite" class.
- Task List:** Empty.
- Failure Trace:** Empty.
- Bottom Status Bar:** Shows "Writable", "Smart Insert", and the current time "19:1:534".

Q4: In this task, we created a JUnit test suite to run tests from Questions 2 and 3 in suite mode. This involved defining a new test class that utilized JUnit annotations to group together the test cases from both programs. By doing so, we were able to execute multiple test programs simultaneously, making it more efficient to run and manage the tests for the Triangle class and the phone number validation class within a single suite.

The screenshot shows the Eclipse IDE interface with the following details:

- Title Bar:** hemalik - Lab1_JUnitAutomatedTesting/src/q5/Fibonacci.java - Eclipse IDE
- Toolbar:** Standard Eclipse toolbar with icons for file operations, search, and project navigation.
- Left Sidebar:** Package Explorer and JUnit view. The JUnit view shows "Finished after 0.015 seconds" with "Runs: 10/10 Errors: 0 Failures: 0".
- Middle Area:** Two code editors: Fibonacci.java and FibonacciTest.java. The Fibonacci.java code is as follows:

```
1 package q5;
2
3 public class Fibonacci {
4
5     // Method to compute the nth Fibonacci number
6     public int compute (int n) {
7         // Base case: If n is 0 or 1, return n (Fibonacci(0) = 0, Fibonacci(1) = 1)
8         int result = 0;
9         if (n <= 1) {
10             result = n;
11         } else {
12             // Recursive case: Fibonacci(n) = Fibonacci(n-1) + Fibonacci(n-2)
13             result = compute(n-1) + compute(n-2);
14         }
15         return result;
16     }
17 }
```

- Right Sidebar:** Task List, Outline, and Problems view. The Outline view shows the class structure: q5 > Fibonacci > compute(int) : int.
- Bottom Status Bar:** Writable, Smart Insert, 18:1:391, and various tool icons.

Q5: In this task, we implemented a JUnit testing program for a given Fibonacci sequence generator. The `compute(int n)` method recursively calculates the Fibonacci number for a given index `n`, with base cases for `n ≤ 1`. To validate its correctness, I created a parameterized JUnit test that checks the computed Fibonacci numbers for indices 0 to 9. Using parameterized testing with a constructor, we passed both the input index and the expected Fibonacci value, ensuring that the method correctly produces the expected sequence while avoiding redundant test cases. This approach streamlined the testing process and effectively verified the accuracy of the Fibonacci computation.

The screenshot shows the Eclipse IDE interface with the following details:

- Title Bar:** hemalik - Lab1_JUnitAutomatedTesting/src/q6/PrimeNumberCheckerTest.java - Eclipse IDE
- Toolbar:** Standard Eclipse toolbar with icons for file operations, search, and project navigation.
- Left Sidebar:** Package Explorer, JUnit (with 5/5 runs, 0 errors, 0 failures), and a Failure Trace section.
- Middle Area:** Two code editors side-by-side:
 - PrimeNumberChecker.java:** A simple class with a static method `isPrime` that returns true if the input is 2 or 3, and false otherwise.
 - PrimeNumberCheckerTest.java:** A JUnit test class with parameterized tests for numbers 2, 6, 19, 22, and 23, comparing the output of the PrimeNumberChecker class against expected boolean values.
- Right Sidebar:** Task List, Outline (showing the structure of PrimeNumberCheckerTest), Problems, Declaration, Console, PIT Mutations, and PIT Summary.
- Status Bar:** Shows the current time as Tue 28 Jan, 23:07, and the date as <terminated> PrimeNumberCheckerTest [JUnit] /usr/local/Eclipse-JavaSDK-2023-12-Rx64/plugins/org.eclipse.jdt.core/jre.full.linux.x86_64_17.0.

Q6: In this task, we implemented a PrimeNumberChecker class to determine whether a given integer is prime. The method returns **true** if the number is prime and **false** otherwise. To validate its functionality, I created a parameterized JUnit test that checks whether the numbers 2, 6, 19, 22, and 23 are prime. Using parameterized testing with a constructor, I passed each number along with its expected result, ensuring that the method correctly identifies prime and non-prime numbers. This approach made the testing process efficient and systematic, verifying the accuracy of the PrimeNumberChecker implementation.

The screenshot shows the Eclipse IDE interface with the following details:

- Title Bar:** hemalik - Lab1_JUnitAutomatedTesting/src/q7/MathematicalTheoryTest.java - Eclipse IDE
- Toolbar:** Standard Eclipse toolbar with icons for file operations, search, and project navigation.
- Left Sidebar:** Package Explorer, JUnit (showing 2/2 runs, 0 errors, 0 failures), and a Failure Trace section.
- Central Area:** Code editor displaying `MathematicalTheoryTest.java`:


```

1 package q7;
2
3 import static org.junit.Assert.assertTrue;
4
5 import org.junit.Assume;
6 import org.junit.experimental.theories.DataPoints;
7 import org.junit.experimental.theories.Theories;
8 import org.junit.experimental.theories.Theory;
9 import org.junit.runner.RunWith;
10
11 @RunWith(Theories.class)
12 public class MathematicalTheoryTest {
13
14     // Provides a set of integers for test cases
15     @DataPoints
16     public static int[] val2 () {
17         return new int[] {0, -1, -10, -1234, 1, 10, 6789};
18     }
19
20     // Test for checking if the sum of a and b is greater than both a and b
21     @Theory
22     public void theo1(Integer a, Integer b) {
23         // Skip test if either a or b is not greater than 0
24         Assume.assumeTrue(a > 0 && b > 0);
25
26         // Assert that sum of a and b is greater than a and b
      
```
- Right Sidebar:** Task List, Outline (showing methods `val2()`, `theo1(Integer, Integer)`, and `theo2(Integer, Integer)`), and Problems view.
- Bottom Status Bar:** Writable, Smart Insert, 37 : 1 : 1032.

Q7: In this task, we implemented JUnit Theories to test the mathematical properties $a + b > a$ and $a + b > b$ for different input values. Theories allow us to test multiple data points dynamically rather than writing individual test cases. If the program assumes both **a** and **b** are positive, pairs like $(0, 0)$, $(-1, -1)$, and $(-10, -1234)$ fail due to negative or zero values, while valid pairs like $(1, 10)$ or $(6789, 6789)$ successfully pass. This approach efficiently verifies the correctness of the mathematical relationships across a range of values while identifying edge cases where the conditions do not hold.

hemalik - Lab1_JUnitAutomatedTesting/src/q8/q5JUnitTheories.java - Eclipse IDE

```

1 package q8;
2 import q5.Fibonacci;
3
4 import static org.junit.Assert.assertTrue;
5
6 import org.junit.Before;
7 import org.junit.experimental.theories.DataPoints;
8 import org.junit.experimental.theories.Theories;
9 import org.junit.experimental.theories.Theory;
10 import org.junit.runner.RunWith;
11
12 @RunWith(Theories.class)
13 public class q5JUnitTheories {
14     // Fibonacci instance to be used in tests
15     Fibonacci fib;
16
17     // Initializes the Fibonacci instance before each test
18     @Before
19     public void init () {
20         fib = new Fibonacci();
21     }
22
23     // Provides pairs of input and expected Fibonacci results for test cases
24     @DataPoints
25     public static int[][] val () {
26         return new int[][] {{0, 0}, {1, 1}, {2, 1}, {3, 2}, {4, 3}, {5, 5}, {6, 8}, {7, 13}, {8, 21}, {9, 34}, {10, 55}, {11, 89}, {12, 144}, {13, 233}, {14, 377}, {15, 610}, {16, 987}, {17, 1597}, {18, 2584}, {19, 4181}, {20, 6774}, {21, 10946}, {22, 17711}, {23, 28657}, {24, 46368}, {25, 75025}, {26, 121393}, {27, 196418}, {28, 317811}, {29, 514229}, {30, 832433}, {31, 1343241}, {32, 2175794}, {33, 3519045}, {34, 5634839}, {35, 9153984}, {36, 14787873}, {37, 24937757}, {38, 40000000}, {39, 64000000}, {40, 100000000}, {41, 164000000}, {42, 264000000}, {43, 428000000}, {44, 700000000}, {45, 1128000000}, {46, 1828000000}, {47, 2956000000}, {48, 4784000000}, {49, 7560000000}, {50, 12320000000}, {51, 20160000000}, {52, 32240000000}, {53, 52400000000}, {54, 84840000000}, {55, 137280000000}, {56, 224000000000}, {57, 361280000000}, {58, 585600000000}, {59, 947200000000}, {60, 1572800000000}, {61, 2545600000000}, {62, 4121600000000}, {63, 6667200000000}, {64, 10800000000000}, {65, 17667200000000}, {66, 28464000000000}, {67, 46131200000000}, {68, 74265600000000}, {69, 120436800000000}, {70, 194697600000000}, {71, 315134400000000}, {72, 510331200000000}, {73, 825462400000000}, {74, 1335793600000000}, {75, 2161256000000000}, {76, 3517051200000000}, {77, 5634080000000000}, {78, 9151131200000000}, {79, 14782251200000000}, {80, 24934371200000000}, {81, 40000000000000000}, {82, 64000000000000000}, {83, 100000000000000000}, {84, 164000000000000000}, {85, 264000000000000000}, {86, 428000000000000000}, {87, 700000000000000000}, {88, 1128000000000000000}, {89, 1828000000000000000}, {90, 3224000000000000000}, {91, 5240000000000000000}, {92, 8484000000000000000}, {93, 13728000000000000000}, {94, 22400000000000000000}, {95, 36128000000000000000}, {96, 58560000000000000000}, {97, 94720000000000000000}, {98, 15728000000000000000}, {99, 25456000000000000000}, {100, 41216000000000000000}, {101, 66672000000000000000}, {102, 108000000000000000000}, {103, 176672000000000000000}, {104, 284640000000000000000}, {105, 461312000000000000000}, {106, 742656000000000000000}, {107, 1204368000000000000000}, {108, 1946976000000000000000}, {109, 3151344000000000000000}, {110, 5103312000000000000000}, {111, 8254624000000000000000}, {112, 1335793600000000000000}, {113, 2161256000000000000000}, {114, 3517051200000000000000}, {115, 5634080000000000000000}, {116, 9151131200000000000000}, {117, 1478225120000000000000}, {118, 2493437120000000000000}, {119, 4000000000000000000000}, {120, 6400000000000000000000}, {121, 10000000000000000000000}, {122, 16400000000000000000000}, {123, 26400000000000000000000}, {124, 42800000000000000000000}, {125, 70000000000000000000000}, {126, 11280000000000000000000}, {127, 18280000000000000000000}, {128, 32240000000000000000000}, {129, 52400000000000000000000}, {130, 84840000000000000000000}, {131, 13728000000000000000000}, {132, 22400000000000000000000}, {133, 36128000000000000000000}, {134, 58560000000000000000000}, {135, 94720000000000000000000}, {136, 15728000000000000000000}, {137, 25456000000000000000000}, {138, 41216000000000000000000}, {139, 66672000000000000000000}, {140, 10800000000000000000000}, {141, 17667200000000000000000}, {142, 28464000000000000000000}, {143, 46131200000000000000000}, {144, 74265600000000000000000}, {145, 12043680000000000000000}, {146, 19469760000000000000000}, {147, 31513440000000000000000}, {148, 51033120000000000000000}, {149, 82546240000000000000000}, {150, 13357936000000000000000}, {151, 21612560000000000000000}, {152, 35170512000000000000000}, {153, 56340800000000000000000}, {154, 91511312000000000000000}, {155, 14782251200000000000000}, {156, 24934371200000000000000}, {157, 40000000000000000000000}, {158, 64000000000000000000000}, {159, 10000000000000000000000}, {160, 16400000000000000000000}, {161, 26400000000000000000000}, {162, 42800000000000000000000}, {163, 70000000000000000000000}, {164, 11280000000000000000000}, {165, 18280000000000000000000}, {166, 32240000000000000000000}, {167, 52400000000000000000000}, {168, 84840000000000000000000}, {169, 13728000000000000000000}, {170, 22400000000000000000000}, {171, 36128000000000000000000}, {172, 58560000000000000000000}, {173, 94720000000000000000000}, {174, 15728000000000000000000}, {175, 25456000000000000000000}, {176, 41216000000000000000000}, {177, 66672000000000000000000}, {178, 10800000000000000000000}, {179, 17667200000000000000000}, {180, 28464000000000000000000}, {181, 46131200000000000000000}, {182, 74265600000000000000000}, {183, 12043680000000000000000}, {184, 19469760000000000000000}, {185, 31513440000000000000000}, {186, 51033120000000000000000}, {187, 82546240000000000000000}, {188, 13357936000000000000000}, {189, 21612560000000000000000}, {190, 35170512000000000000000}, {191, 56340800000000000000000}, {192, 91511312000000000000000}, {193, 14782251200000000000000}, {194, 24934371200000000000000}, {195, 40000000000000000000000}, {196, 64000000000000000000000}, {197, 10000000000000000000000}, {198, 16400000000000000000000}, {199, 26400000000000000000000}, {200, 42800000000000000000000}, {201, 70000000000000000000000}, {202, 11280000000000000000000}, {203, 18280000000000000000000}, {204, 32240000000000000000000}, {205, 52400000000000000000000}, {206, 84840000000000000000000}, {207, 13728000000000000000000}, {208, 22400000000000000000000}, {209, 36128000000000000000000}, {210, 58560000000000000000000}, {211, 94720000000000000000000}, {212, 15728000000000000000000}, {213, 25456000000000000000000}, {214, 41216000000000000000000}, {215, 66672000000000000000000}, {216, 10800000000000000000000}, {217, 17667200000000000000000}, {218, 28464000000000000000000}, {219, 46131200000000000000000}, {220, 74265600000000000000000}, {221, 12043680000000000000000}, {222, 19469760000000000000000}, {223, 31513440000000000000000}, {224, 51033120000000000000000}, {225, 82546240000000000000000}, {226, 13357936000000000000000}, {227, 21612560000000000000000}, {228, 35170512000000000000000}, {229, 56340800000000000000000}, {230, 91511312000000000000000}, {231, 14782251200000000000000}, {232, 24934371200000000000000}, {233, 40000000000000000000000}, {234, 64000000000000000000000}, {235, 10000000000000000000000}, {236, 16400000000000000000000}, {237, 26400000000000000000000}, {238, 42800000000000000000000}, {239, 70000000000000000000000}, {240, 11280000000000000000000}, {241, 18280000000000000000000}, {242, 32240000000000000000000}, {243, 52400000000000000000000}, {244, 84840000000000000000000}, {245, 13728000000000000000000}, {246, 22400000000000000000000}, {247, 36128000000000000000000}, {248, 58560000000000000000000}, {249, 94720000000000000000000}, {250, 15728000000000000000000}, {251, 25456000000000000000000}, {252, 41216000000000000000000}, {253, 66672000000000000000000}, {254, 10800000000000000000000}, {255, 17667200000000000000000}, {256, 28464000000000000000000}, {257, 46131200000000000000000}, {258, 74265600000000000000000}, {259, 12043680000000000000000}, {260, 19469760000000000000000}, {261, 31513440000000000000000}, {262, 51033120000000000000000}, {263, 82546240000000000000000}, {264, 13357936000000000000000}, {265, 21612560000000000000000}, {266, 35170512000000000000000}, {267, 56340800000000000000000}, {268, 91511312000000000000000}, {269, 14782251200000000000000}, {270, 24934371200000000000000}, {271, 40000000000000000000000}, {272, 64000000000000000000000}, {273, 10000000000000000000000}, {274, 16400000000000000000000}, {275, 26400000000000000000000}, {276, 42800000000000000000000}, {277, 70000000000000000000000}, {278, 11280000000000000000000}, {279, 18280000000000000000000}, {280, 32240000000000000000000}, {281, 52400000000000000000000}, {282, 84840000000000000000000}, {283, 13728000000000000000000}, {284, 22400000000000000000000}, {285, 36128000000000000000000}, {286, 58560000000000000000000}, {287, 94720000000000000000000}, {288, 15728000000000000000000}, {289, 25456000000000000000000}, {290, 41216000000000000000000}, {291, 66672000000000000000000}, {292, 10800000000000000000000}, {293, 17667200000000000000000}, {294, 28464000000000000000000}, {295, 46131200000000000000000}, {296, 74265600000000000000000}, {297, 12043680000000000000000}, {298, 19469760000000000000000}, {299, 31513440000000000000000}, {300, 51033120000000000000000}, {301, 82546240000000000000000}, {302, 13357936000000000000000}, {303, 21612560000000000000000}, {304, 35170512000000000000000}, {305, 56340800000000000000000}, {306, 91511312000000000000000}, {307, 14782251200000000000000}, {308, 24934371200000000000000}, {309, 40000000000000000000000}, {310, 64000000000000000000000}, {311, 10000000000000000000000}, {312, 16400000000000000000000}, {313, 26400000000000000000000}, {314, 42800000000000000000000}, {315, 70000000000000000000000}, {316, 11280000000000000000000}, {317, 18280000000000000000000}, {318, 32240000000000000000000}, {319, 52400000000000000000000}, {320, 84840000000000000000000}, {321, 13728000000000000000000}, {322, 22400000000000000000000}, {323, 36128000000000000000000}, {324, 58560000000000000000000}, {325, 94720000000000000000000}, {326, 15728000000000000000000}, {327, 25456000000000000000000}, {328, 41216000000000000000000}, {329, 66672000000000000000000}, {330, 10800000000000000000000}, {331, 17667200000000000000000}, {332, 28464000000000000000000}, {333, 46131200000000000000000}, {334, 74265600000000000000000}, {335, 12043680000000000000000}, {336, 19469760000000000000000}, {337, 31513440000000000000000}, {338, 51033120000000000000000}, {339, 82546240000000000000000}, {340, 13357936000000000000000}, {341, 21612560000000000000000}, {342, 35170512000000000000000}, {343, 56340800000000000000000}, {344, 91511312000000000000000}, {345, 14782251200000000000000}, {346, 24934371200000000000000}, {347, 40000000000000000000000}, {348, 64000000000000000000000}, {349, 10000000000000000000000}, {350, 16400000000000000000000}, {351, 26400000000000000000000}, {352, 42800000000000000000000}, {353, 70000000000000000000000}, {354, 11280000000000000000000}, {355, 18280000000000000000000}, {356, 32240000000000000000000}, {357, 52400000000000000000000}, {358, 84840000000000000000000}, {359, 13728000000000000000000}, {360, 22400000000000000000000}, {361, 36128000000000000000000}, {362, 58560000000000000000000}, {363, 94720000000000000000000}, {364, 15728000000000000000000}, {365, 25456000000000000000000}, {366, 41216000000000000000000}, {367, 66672000000000000000000}, {368, 10800000000000000000000}, {369, 17667200000000000000000}, {370, 28464000000000000000000}, {371, 46131200000000000000000}, {372, 74265600000000000000000}, {373, 12043680000000000000000}, {374, 19469760000000000000000}, {375, 31513440000000000000000}, {376, 51033120000000000000000}, {377, 82546240000000000000000}, {378, 13357936000000000000000}, {379, 21612560000000000000000}, {380, 35170512000000000000000}, {381, 56340800000000000000000}, {382, 91511312000000000000000}, {383, 14782251200000000000000}, {384, 24934371200000000000000}, {385, 40000000000000000000000}, {386, 64000000000000000000000}, {387, 10000000000000000000000}, {388, 16400000000000000000000}, {389, 26400000000000000000000}, {390, 42800000000000000000000}, {391, 70000000000000000000000}, {392, 11280000000000000000000}, {393, 18280000000000000000000}, {394, 32240000000000000000000}, {395, 52400000000000000000000}, {396, 84840000000000000000000}, {397, 13728000000000000000000}, {398, 22400000000000000000000}, {399, 36128000000000000000000}, {400, 58560000000000000000000}, {401, 94720000000000000000000}, {402, 15728000000000000000000}, {403, 25456000000000000000000}, {404, 41216000000000000000000}, {405, 66672000000000000000000}, {406, 10800000000000000000000}, {407, 17667200000000000000000}, {408, 28464000000000000000000}, {409, 46131200000000000000000}, {410, 74265600000000000000000}, {411, 12043680000000000000000}, {412, 19469760000000000000000}, {413, 31513440000000000000000}, {414, 51033120000000000000000}, {415, 82546240000000000000000}, {416, 13357936000000000000000}, {417, 21612560000000000
```

possible input combinations. By applying theory-based testing, we made the tests more dynamic, efficient, and flexible, ensuring the correctness of both the Fibonacci sequence calculator and the Prime Number Checker across a wider range of values without manually defining individual test cases.