21110324_LuongDangKhoi_KTPM Lab03

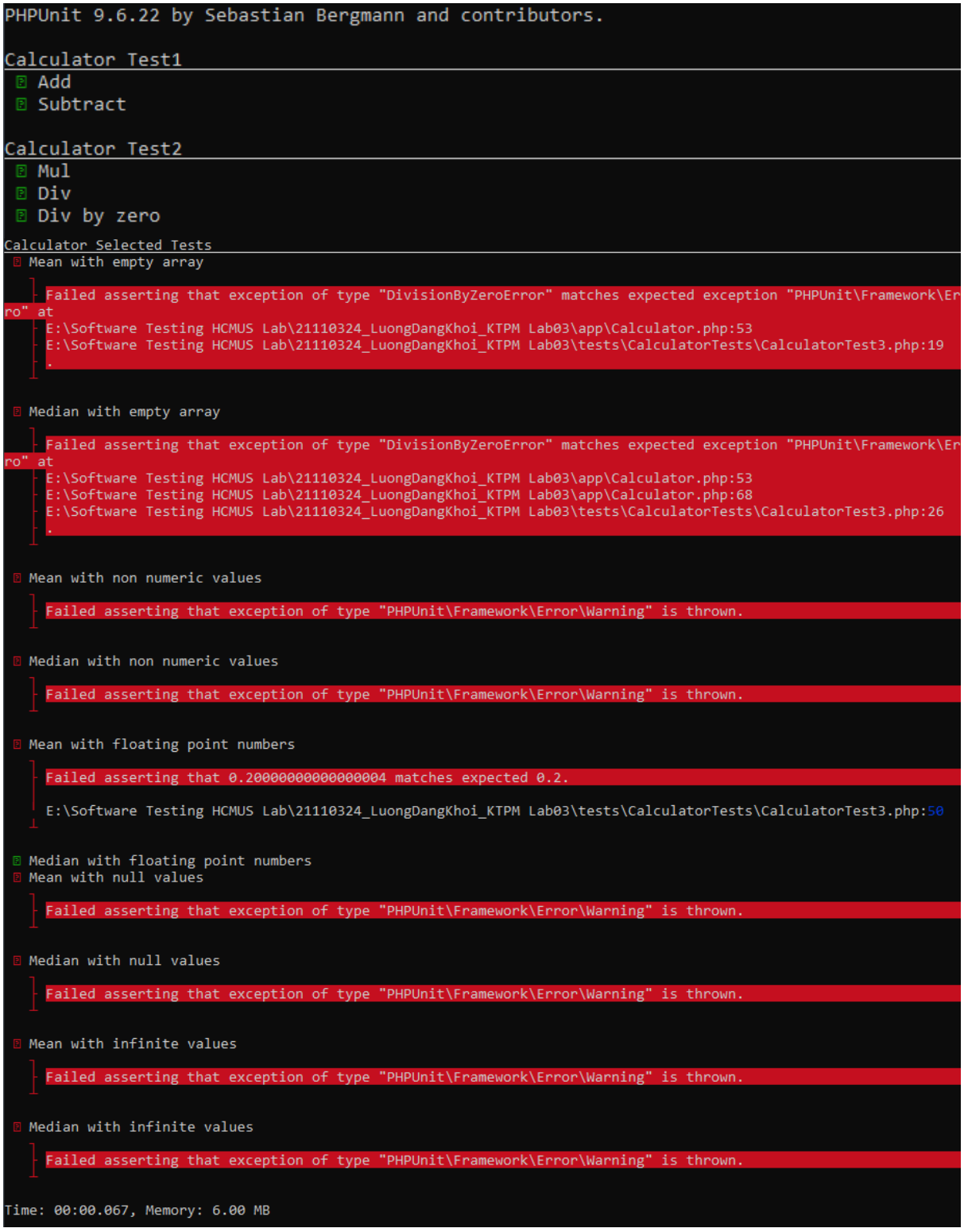
*** Calculator Testing:**

Các nội dung testing:

CalculatorTest1	CalculatorTest2	CalculatorTest3
- Add - Subtract	- Mul - Div - Div by zero	- Mean with empty array - Median with empty array - Mean with non-numberic values - Median with non-numberic values - Mean with floating point numbers - Median with floating point numbers - Mean with null values - Median with null values - Mean with infinite values - Median with infinite values

Kết quả:

Testcases	Result
Add	Pass
Subtract	Pass
Mul	Pass
Div	Pass
Dib by zero	Pass
Mean with empty array	Fail
Median with empty array	Fail
Mean with non-numberic values	Fail
Median with non-numberic values	Fail
Mean with floating point numbers	Fail
Median with floating point numbers	Pass
Mean with null values	Fail
Median with null values	Fail
Mean with infinite values	Fail
Median with infinite values	Fail



Sửa lỗi: Kiểm tra đầu vào null – infinite – string – invalid types và áp dụng cho cả các function khác trong class

-- Code cuối file --

*** AdvancedMath Testing:**

AdvancedMath.php có các functions: giai thừa, luỹ thừa, căn bậc 2, tổng số chẵn trong mảng, kiểm tra số nguyên tố, tìm số lớn nhất trong mảng.

Factorial	Power	Square	Evens Sum	Prime Check	Find Max
- Positive - Zero - Negative - Large - String - Null - Infinite - Array	- Positive - Zero base - Zero exponent - Zero - Negative base - Negative exponent - Large - String - Null - Infinite base - Infinite exponent - Array	- Positive - Zero - Negative - Large - String - Null - Infinite - Array	- Positive - Mixed - All odd - Empty - Negative - String - Zero - Infinite - Non-array		

Kết quả: Lược bỏ ảnh chụp terminal để ngắn gọn

	Positive	Zero	Negative	Large	String	Null	Infinite	Array
factorial	Pass	Pass	Pass	Fail	Fail	Fail	Fail	Fail

	power
Positive	Pass
Zero base	Pass
Zero exponent	Pass
Zero	Pass
Negative base	Pass
Negative exponent	Pass
Large	Fail
String	Fail
Null	Fail
Infinite base	Fail
Infinite exponent	Fail
Array	Fail

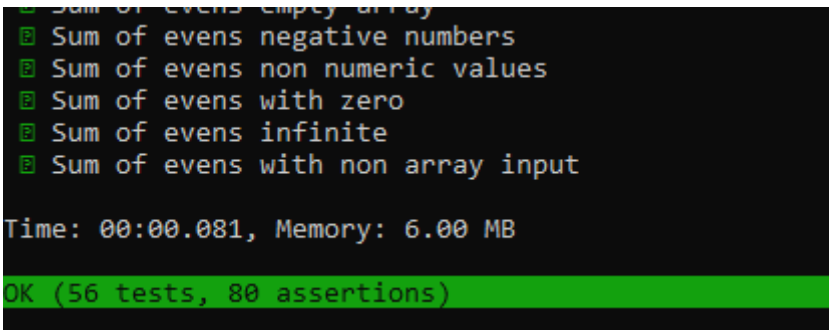
	squareRoot
Positive	Pass
Zero	Pass
Negative	Pass
Large	Pass
String	Fail
Null	Fail
Infinite	Fail
Array	Fail

	sonOfEvens
Positive	Pass
Mixed	Pass
All odd	Pass
Empty	Pass
Negative	Pass
String	Fail
Zero	Pass
Infinite	Fail
Non-array	Fail

	isPrime
Positive	Pass
Posive non-Prime	Pass
Zero	Pass
Negative	Pass
Large	Pass
String	Fail
Null	Fail
Infinite	Fail
Array	Fail

	findMax
Positive	Pass
Negative	Pass
Mixed	Pass
Single element	Pass
Empty	Pass
Duplicate max	Pass
Non-array input	Fail
Non-numberic	Fail
Infinite	Pass
Zero	Pass

*** Sửa lỗi cho AdvancedMath:**



<?php

namespace App;

```
class AdvancedMath
{
    // Tính giai thừa của một số
    public function factorial($number)
    {
        if (!is_int($number)) {
            throw new \InvalidArgumentException("Invalid input type");
        }

        if ($number < 0) {
            throw new \InvalidArgumentException("Factorial is not defined for negative
numbers");
        }

        if ($number === INF) {
            throw new \InvalidArgumentException("Invalid input type");
        }

        if ($number > 170) { // Beyond this, PHP's `gmp_fact()` can't handle the result
            throw new \OverflowException("Factorial result too large");
        }

        $result = 1;
        for ($i = 1; $i <= $number; $i++) {
            $result *= $i;
        }

        return $result;
    }

    // Tính lũy thừa
    public function power($base, $exponent)
    {
```

```
    if (!is_numeric($base) || !is_numeric($exponent)) {
        throw new \InvalidArgumentException("Invalid input type");
    }

    if (is_infinite($base) || is_infinite($exponent)) {
        throw new \InvalidArgumentException("Invalid input type");
    }

    try {
        $result = $base ** $exponent;
    } catch (\ArithmeticError $e) {
        throw new \OverflowException("Result too large");
    }

    if (is_infinite($result)) {
        throw new \OverflowException("Result too large");
    }

    return $result;
}


// Tính căn bậc hai
public function squareRoot($number)
{
    if (!is_numeric($number)) {
        throw new \InvalidArgumentException("Invalid input type");
    }

    if ($number < 0) {
        return null;
    }
```

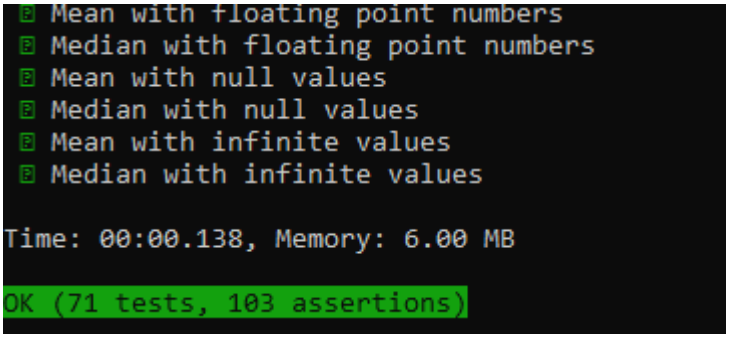
```
    if (is_infinite($number)) {
        throw new \InvalidArgumentException("Infinite is invalid");
    }

    return sqrt($number);
}

// Tính tổng của các số chẵn trong một mảng
public function sumOfEvens($numbers)
{
    if (!is_array($numbers)) {
        throw new \InvalidArgumentException("Input must be an array");
    }

    foreach ($numbers as $item) {
        if (!is_numeric($item)) {
            throw new \InvalidArgumentException("All elements must be numbers");
        }

        if (is_infinite($item)) {
            throw new \InvalidArgumentException("Infinite values are not allowed");
        }
    }

    return array_reduce($numbers, function ($carry, $item) {
        if ($item % 2 == 0) {
            $carry += $item;
        }
        return $carry;
    }, 0);
}
```

```
// Kiểm tra số nguyên tố
public function isPrime($number)
{
    if ($number === INF || $number === -INF) {
        throw new \InvalidArgumentException("Infinite values are not allowed");
    }

    if (!is_int($number)) {
        throw new \InvalidArgumentException("Input must be an integer");
    }

    if ($number < 2) {
        return false;
    }

    for ($i = 2; $i <= sqrt($number); $i++) {
        if ($number % $i == 0) {
            return false;
        }
    }

    return true;
}

// Tìm số lớn nhất trong một mảng
public function findMax(array $numbers)
{
    if (empty($numbers)) {
        return null;
    }

    foreach ($numbers as $num) {
```

```
            if (!is_numeric($num)) {
                throw new \InvalidArgumentException("All elements must be numeric");
            }
        }

        return max($numbers);
    }
}
```

==========
==========
==========
==========
==========

*** Sửa lỗi cho Calculator:**




```
<?php

namespace App;

class Calculator
{
    // Constants for rounding precision
    const ROUND_PRECISION = 2;

    /**
     * Calculate the summary
     * @param $num1, num2 numbers
     * @return float summary result
     */
    public function add($num1, $num2)
    {
        // Check for INF
        if (is_infinite($num1) || is_infinite($num2)) {
            throw new \InvalidArgumentException("One or both values are infinite");
        }
```

```
        // Check for invalid input type (array instead of number)
        if (is_array($num1) || is_array($num2) || !is_numeric($num1) || !is_numeric($num2)) {
            throw new \InvalidArgumentException("Both inputs must be numbers, not arrays or non-numeric values");
        }

        return $this->roundResult($num1 + $num2);
    }

    /**
     * Calculate the difference
     * @param $num1, num2 numbers
     * @return float difference result
     */
    public function subtract($num1, $num2)
    {
        // Check for INF
        if (is_infinite($num1)) {
            throw new \InvalidArgumentException("num1 cannot be infinite");
        }

        if (is_infinite($num2)) {
            throw new \InvalidArgumentException("num2 cannot be infinite");
        }

        if (is_infinite($num1) && is_infinite($num2)) {
            throw new \InvalidArgumentException("Both numbers cannot be infinite");
        }

        // Check for invalid input type (array instead of number)
        if (is_array($num1) || is_array($num2) || !is_numeric($num1) || !is_numeric($num2)) {
```

```
            throw new \InvalidArgumentException("Both inputs must be numbers, not arrays or non-numeric values");
        }

        return $this->roundResult($num1 - $num2);
    }

    /**
     * Calculate the multiplication
     * @param $num1, num2 numbers
     * @return float multiplication result
     */
    public function multiply($num1, $num2)
    {
        // Check for INF
        if (is_infinite($num1) || is_infinite($num2)) {
            throw new \InvalidArgumentException("One or both values are infinite");
        }

        // Check for invalid input type (array instead of number)
        if (is_array($num1) || is_array($num2) || !is_numeric($num1) || !is_numeric($num2)) {
            throw new \InvalidArgumentException("Both inputs must be numbers, not arrays or non-numeric values");
        }

        return $this->roundResult($num1 * $num2);
    }

    /**
     * Calculate the division
     * @param $num1, num2 numbers
     * @return float division result
     */
```

```
    public function divide($num1, $num2)
    {
        // Check for INF
        if (is_infinite($num1) || is_infinite($num2)) {
            throw new \InvalidArgumentException("One or both values are infinite");
        }

        // Check for zero division
        if ($num2 == 0) {
            throw new \InvalidArgumentException("Division by zero");
        }

        // Check for invalid input type (array instead of number)
        if (is_array($num1) || is_array($num2) || !is_numeric($num1) || !is_numeric($num2)) {
            throw new \InvalidArgumentException("Both inputs must be numbers, not arrays or
non-numeric values");
        }

        return $this->roundResult($num1 / $num2);
    }

    /**
     * Calculate the mean average
     * @param array $numbers Array of numbers
     * @return float Mean average
     */
    public function mean(array $numbers)
    {
        if (empty($numbers)) {
            throw new \InvalidArgumentException("Array cannot be empty");
        }

        // Check for invalid input type (array elements must be numbers)
```

```
    foreach ($numbers as $number) {
        if (!is_numeric($number)) {
            throw new \InvalidArgumentException("Array elements must be numeric values");
        }
    }

    // Check for INF in the array
    foreach ($numbers as $number) {
        if (is_infinite($number)) {
            throw new \InvalidArgumentException("Array cannot contain INF");
        }
    }

    // Calculate and round the result
    $result = array_sum($numbers) / count($numbers);
    return $this->roundResult($result);
}

/**
 * Calculate the median average
 * @param array $numbers Array of numbers
 * @return float Median average
 */
public function median(array $numbers)
{
    if (empty($numbers)) {
        throw new \InvalidArgumentException("Array cannot be empty");
    }

    // Check for invalid input type (array elements must be numbers)
    foreach ($numbers as $number) {
        if (!is_numeric($number)) {
```

```
                throw new \InvalidArgumentException("Array elements must be numeric values");
            }
        }

        // Check for INF in the array
        foreach ($numbers as $number) {
            if (is_infinite($number)) {
                throw new \InvalidArgumentException("Array cannot contain INF");
            }
        }

        sort($numbers);
        $size = count($numbers);
        if ($size % 2) {
            return $this->roundResult($numbers[$size / 2]);
        } else {
            return $this->roundResult($this->mean(
                array_slice($numbers, ($size / 2) - 1, 2)
            ));
        }
    }

    /**
     * Rounds the result to the defined precision
     * @param float $result The calculated result
     * @return float The rounded result
     */
    private function roundResult($result)
    {
        return round($result, self::ROUND_PRECISION);
    }
}
```