Scientific Calculator

Course - SOEN 6011, Professor - Pankaj Kamthan

Shagun Shagun, ID - 40138455

1 Problem 1

1.1 Description

A power function is of the form:

$$f(x) = ab^x \tag{1}$$

where x is a real number, a and b are constants.

1.2 Domain

The domain is set of all real numbers, (- ∞,∞)

1.3 Co-domain

The co-domain is also set of all real numbers.

1.4 Characteristics of Power Function.

1. For any exponential function, the domain is the set of all real number, however range is bounded by the horizontal symptote of the graph of f(x).
2. The behaviour of power function effects the exponential growth and decay.
3. When b greater than 1, the graph accelerates towards y-axis contributing to exponential growth.
4. When b greater than 1 and less than 0, the graph decreases towards y-axis contributing to exponential decay.
5. When modeling real-world situations with an exponential function, the domain and range can be limited to numbers that make sense in the context. The domain and range can be stated using the inequalities for a continuous interval in these cases.

2 Problem 2

2.1 Assumptions

- In function $f(x) = ab^x$, the output will always be in decimals.
- The output of the function will be greater than zero, if a and b constants are not equal to zero.
- The range for the values of x can be from -100000 $\leq$ x $\leq$ +100000.
- x can be negative but not in decimal.
- The value for a and b ranging from -100000 $\leq$ a $\leq$ 100000 and -100000 $\leq$ b $\leq$ 100000.
- a and b are constants, the calculator accepts the magical constants such as e.

2.2 Requirements

1. **First Requirement**
   - **ID =** FR1
   - **Type =** Functional Requirements
   - **Version =** 1.0
   - **Difficulty =** Easy
   - **Priority =** 1
   - **Description =** System shall take an input x as a real number.
   - **Rationale =** The rationale behind this requirement is to calculate the function only for real numbers.
2. **Second Requirements**
   - **ID =** FR2
   - **Type =** Functional Requirements
   - **Version =** 1.0
   - **Difficulty =** Easy
   - **Priority =** 1
   - **Description =** System should validate the input.
   - **Rationale =** The rationale behind this requirement is to check domain value for function.
3. **Third Requirement**
   - **ID =** FR3
   - **Type =** Functional Requirements

- **Version =** 1.0
- **Difficulty =** Easy
- **Priority =** 3
- **Description =** System shall output the value within the expected range.
- **Rationale =** The rationale behind this requirement is to get result in the range of the function.

4. **Fourth Requirement**

   - **ID =** FR4
   - **Type =** Functional Requirements
   - **Version =** 1.0
   - **Difficulty =** Easy
   - **Priority =** 2
   - **Description =** System shall give a input the value of x within given range.
   - **Rationale =** The rationale behind this requirement is to get rational output of a function each time.

5. **Fifth Requirement**

   - **ID =** FR5
   - **Type =** Functional Requirements
   - **Version =** 1.0
   - **Difficulty =** Easy
   - **Priority =** 1
   - **Description =** System shall show relevant error messages if any.
   - **Rationale =** The rationale behind this requirement is to handle error handling.

6. **Sixth Requirement**

   - **ID =** FR6
   - **Type =** Functional Requirements
   - **Version =** 1.0
   - **Difficulty =** Medium
   - **Priority =** 1
   - **Description =** System shall accepts magical constant e as constant.
   - **Rationale =** The rationale behind this requirement is the acceptability of the constants.

7. **Seventh Requirement**

- **ID =** FR7
- **Type =** Non-Functional Requirements
- **Version =** 1.0
- **Difficulty =** Easy
- **Priority =** 1
- **Description =** System shall show relevant success messages or confirmation results.
- **Rationale =** The rationale behind this requirement is to show relevant messages and contributes to usability.

8. **Eighth Requirement**

   - **ID =** FR8
   - **Type =** Non-Functional Requirements
   - **Version =** 1.0
   - **Difficulty =** Easy
   - **Priority =** 2
   - **Description =** System shall complete the calculation on expected time.
   - **Rationale =** The rationale behind this requirement is to have good performance of the calculator.

3 Problem 3

3.1 Algorithm and Pseudocode

Following is algorithm and the pseudo-code for ab^x

3.2 Algorithm Description

3.2.1 Algorithm 1

The details of algorithm 1 is given following:
Complexity = $O(logn), \Omega(n)$
Space Complexity = $O(1)$
Approach = Recursion
Rationale = The a and b are the constants, depending on the value of b the complexities is being calculated such that if b = any integer then f5Power() will be called. So, the time complexity will be $O(logn)$ and space complexity to 1. If b = e then taylor series will execute calling exponential() resulting $\Omega(n)$ time complexity and 1 space complexity.
Advantages

1. $O(logn)$ is the time complexity of the recursive algorithm. This recursive technique has tail recursive to avoid stack overflow issues and handle large inputs more efficiently.

2. Algorithm 1 has more readability and maintainable code.

Algorithm 1 Recursive Approach - ab^x

```
procedure Function5(a, b, x)
    in:  String a, b, x
    out:  double result
    result = 0
    if ((a || b) == "0") then return result
    else
        if (b == "e") then
            result = a * exponential(x)
        else
            result = a * f5Power(b,x)
        return result;

procedure exponential(n)
    in:  int n
    out:  double result
    sum = 1
    m = 1
    for i <= n
    sum = 1 + m * sum/i
    return sum

procedure f5Power(x, n)
    in:  double x, int n
    out:  double result
    if ((n < 0) then
        return 1.0/powerHandler(x, n)
    else
        return powerHandler(x, n)

procedure powerHandler(x, n)
    in:  double x, double n
    out:  double result
    if (n == 0) then return 1
    if (n == 1) then return x
    if (nmod2 == 0) then
        return powerHandler(x * x, n/2)
    else
        return x * powerHandler(x * x, n/2)
```

Algorithm 2 Iterative Approach - ab^x

```
procedure Function5(a, b, x)
    in: String a, b, x
    out: double result
    result = 0
    if ((a || b) == "0") then return result
    else
        if (b == "e") then
            sum = 1
            m = 1 and i = 1
            for i <= x
            sum = 1 + m * sum/i
            end
            return a*sum
        else
            if (x == 0) then return 1
            if (x == 1) then return a*x
            else
                i = 1andpow = 1
                for i <= n
                pow = pow * x
                end
                return a*sum
```

3.2.2 Algorithm 2

The details of algorithm 2 is given following:
Complexity = $O(n), \Omega(n)$
Space Complexity = $O(1)$
Approach = Iterative
Rationale = The a and b are the constants, depending on the value of b the complexities is being calculated such that if b = any integer then f5Power() will be called. So, the time complexity will be $O(n)$ and space complexity to 1. If b = e then taylor series will execute calling exponential() resulting $\Omega(n)$ time complexity and 1 space complexity.
Advantages

1. Because all operations are performed on the heap, iterative algorithms do not incur from stack overflow.
2. Space complexity is O(1).

Disadvantages

1. The complexity is $O(n)$.
2. The algorithm is not efficient.
3. The algorithm has poor readability and maintainability.

Algorithm 3 Divide and Conquer Approach - ab^x

```
procedure Function5(a, b, x)
    in: String a, b, x
    out: double result
    result = 0
    if ((a || b) == "0") then return result
    else
        if (b == "e") then
            sum = 1
            m = 1 and i = 1
            for i <= x
            sum = 1 + m * sum/i
            end
            return a*sum
        else
            if (x == 0) then return 1
            if (x == 1) then return a*x
            else
                return a*calculatorPower(b,x)

procedure calculatorPower(b, x)
    in: double b, int x
    out: double result
    if (x == 1) then
        return 1
    else if (x mod 2 == 0) then
        return calculatorPower(b, x/2) * calculatorPower(b, x/2)
    else
        return b * calculatorPower(b, x/2) * calculatorPower(b, x/2)
```

3.2.3 Algorithm 3

The details of algorithm 1 is given following:
Complexity = $O(n), \Omega(n)$
Space Complexity = $O(1)$
Approach = Divide and Conquer
Rationale = The a and b are the constants, depending on the value of b the complexities is being calculated such that if b = any integer then f5Power() will be called. So, the time complexity will be $O(n)$ and space complexity to 1. If b = e then taylor series will execute calling exponential() resulting $\Omega(n)$ time complexity and 1 space complexity.
Advantages

1. The problem has been solved in divide and conquer approach may result in robustness.
2. Space complexity is O(1).

Disadvantages

1. The complexity is $O(n)$.
2. The constant allocation of memory space, which leads to a stack overflow, has a substantial impact on efficiency.

3.3 Conclusion

By computing power(b, x/2) only once and storing it, the algorithm 2 and algorithm 3 can be optimized to $O(logn)$. The approach used in algorithm stores power only at once. Algorithm Implementation = Algorithm 1

4 Problem 4

4.1 Eclipse Debugger

Debugging is the process of finding and fixing faults, flaws, and anomalies in software. It's an essential ability for any Java developer because it aids in the detection of minor bugs that aren't obvious during code reviews or only occur when a specific condition is met.

Debugger Used : Inbuilt debugger offered by Eclipse Java Development Tools (JDT).

Following are the advantages and disadvantages of using debugger in the program.

Advantages

- Debugger promptly reports an erroneous state. This allows for earlier fault detection and makes the software development process stress-free and trouble-free.

- Debugging aids the developer in eliminating redundant and unwanted data.
- Debugging allows developers to avoid writing complex one-time testing code, which reduces resources and time during software development.

Disadvantages

- When the execution is halted inside an invariant, the debugger isn't particularly useful.
- If you use any of the previously unsupported expressions in a breakpoint condition, the condition will always return True since the evaluation is failing. In this instance, the debugger will come to a halt.

4.2 Achieved Quality Attributes

4.2.1 Maintainability

- Common practices has been adapted within a group to avoid ambiguities.
- Useful comments added in the code where it is required.
- Avoided global scoping for common variables and functions.
- Refactored code once every member merged their code to the github branch.

4.2.2 Robustness

- Usage of exception for exceptional test cases.
- Narrowed the variable scope as far as possible in the code.
- Error handling done on the code.
- Usage of mutable variable over creating new variables.

4.2.3 Usable

- Simple console interface provided for user input.
- Error messages are given in wrong input.
- Success messages are given for results.
- Suggestions are given when user encountered any difficulties while using calculator.

4.2.4 Correctness

- Coding standards is followed.
- Proper testing practices has been done on the function assigned.
- JUnit Testing has implemented to check the correctness.

4.2.5 Efficiency

- The main focus on readability is given to the code.4
- The program takes not less than two or three nanoseconds.

4.3 Quality Check of Source Code

It's a programming tool that helps programmers write Java code that follows a set of rules. It automates the process of inspecting Java code, saving humans the time and effort of doing so. It's ideal for projects that want to enforce a coding standard.

Advantages

- The checkstyle is portable between different IDEs.
- Easily integrate as a pre-commit hook or into your build tool into your Software Configuration Management.
- Checkstyle is a stand-alone framework, integrating it with your other tools is considerably easier.

Disadvantages

- Checkstyle is a static analysis tool for a single file.

5 Problem 5

5.1 Standard Guidelines

The source code review is based on the standard practices that a developer has used while writing the code. The coverage of all standard guidelines on which the report has generated is listed below.

1. The code should eliminate nested if statement.
2. The code should follow a coding style guide.
3. The code should have meaningful method and variable names.
4. The author should annotate the comments not exceeding 10 to 20 lines.
5. The class program should not import unnecessary packages.
6. The code should have low coupling and high cohesion.
7. The code should avoid the global scoping for common variables and functions.
8. The code should avoid memory leakage.
9. The code should narrow the variable scope as far as possible in the code.
10. Usage of mutable variable over creating new variables in object oriented language.

5.2 Review Approach

Used Approach - Lightweight Code Review

- Walk-through the code.
- Executed the code in run time environment and also used debugger.
- Inspection on comment length, Checking dependencies among the components, Check on unused variables.
- Observing global scoping of the functions and variables.

5.3 Results

No.	Description	Result
1	Use of annotation in the code	PASS
2	No usage of nested ifs statement	PASS
3	No unused variable found	PASS
4	No memory leakage	PASS
5	Usage of meaningful method and variable names	PASS
6	Not exceeding source line of code	PASS
7	No deadlock encountered	PASS
8	No dependency found in the code	PASS
9	No Error found in the code	PASS
9	No Warning found in the code	PASS
10	No unnecessary package is included in the code	PASS

5.4 Conclusion

Overall, the code is well-structured and well-written. It is clear from the code that readability is a developer's primary focus. The only suggestion I have is to retain logs because they can help with debugging.

6 Problem 6

6.1 Standard Guidelines

The following are the guidelines implemented in the unit testing for function 5 :

$$f(x) = ab^x \tag{2}$$

- Each test case should given a test case id.
- Each test should contains the unit test of each function involved in the implementation of function 5.
- Each test case is followed by one line statement justifying it's description.
- Each test case contains the function name used in the unit test case.

- Test case shall cover all the cases expected.
- Test Case should mention a expected value and actual value.
- Atleast one test case should be consider into a delta precision values.
- Each test case should show message or log for the sake of readability.
- Each test case should be traceable.
- Test case should map to the functional requirement.

6.2 Implemented Unit Test

- Unit Testing Framework Used: JUnit
- The sample practice used in the test cases are as follows:

  ID = TC1
  Test Case = description about the test case.
  Function Used = name of the function used in the test case.
  Functional Requirement ID Used = FRn

7 Problem 7

7.1 Testing Function2 - tanx

The test review is based on the test cases developer has mentioned and also the check on the mapping of each test case with requirements mentioned. The coverage of all test cases are included.

7.2 Testing Criteria

The developer has mentioned a set of test cases in which all the functions and implementation has been covered. The following are the points based on which this test cases are evaluated

1. Code Execution.
2. Test Case returning positive responses.
3. Test Cases mapped to functional requirements.
4. Coverage of all test cases.
5. Working of each unit in the program.
6. Covering input validation.

7.3 Results

No.	Description	Result
1	Test Cases covers of all possible cases	PASS
2	Test Cases covers the exception handling	PASS
3	Requirement mentioned in the problem2 has covered	PASS
4	Working of calculating sin() function	PASS
5	Working of calculating cos() function	PASS
6	Working of calculating tan() function	PASS
7	Test on checking input within range	PASS
8	Test cases follows assumptions	Not Applicable

7.4 Conclusion

Almost all the methods of function2 have been tested and confirmed to be effective and corrective. During the testing process, no major mistakes were discovered. All of the requirements given in Problem 2 has been covered in the mentioned test cases. Furthermore, all the test cases are to the point and covered all possibilities.

7.5 Suggestions

Assigning each test case a unique id would make contribution to readability.

References

[1] Wikiedia: Exponent Function,
`https://en.wikipedia.org/wiki/Exponential_function`

[2] Geeksforgeeks: Power Function,
`https://www.geeksforgeeks.org/write-a-c-program-to-calculate-powxn/`