SOEN-6011 Software Engineering Processes

ETERNITY

Function 7: Power Function

Professor: Dr. Pankaj Kamthan
Author: Manjit Singh(40185580)

PROBLEM 1

1 Power Function

1.1 Definition of the Function

A power function is represented as:

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

It is read as x is raised to the power y, where x is the base vale and y is the exponent/power value.

1.2 Domain of the Function

It depends upon the power of the function:-

- If power p is a non-negative integer, the domain is all real numbers: (- ∞,∞).
- If power p is a negative integer, the domain is all real numbers excluding zero ((- ∞, 0) $\cup$ (0,∞))
- If power p is a rational number expressed in lowest terms as a/b and b is even and p $>$ 0, the domain is all non-negative real numbers: (0,∞) and x | x $\geq$ 0.

  If p $<$ 0, The domain is positive real numbers :(0,∞) and x | x $>$ 0.
- If power p is a rational number expressed in lowest terms as a/b and b is odd, p $<$ 0, The domain is all real numbers.

  If p $<$ 0, The domain is all real numbers not including zero.

1.3 Co-Domain of the function

- The co-domain of a function is the set of its possible outputs.
- Co-Domain of power function consists to real numbers.

1.4 Context of Use of Power Function

An analysis of the many circumstances under which a software system is employed in various working environments

Technical Users -

- **Software Developers** - The system's creators utilise it to see how they might increase productivity in terms of speed and accuracy.

- **Testers** -The system is tested to determine whether it functions as intended. They will in this instance determine whether or not the calculator produces highly precise results.
- **Concerned Stakeholders** - These could be the investors who are mainly inclined towards the business profit.
- **Project Manager** - They are concerned whether everything is being developed properly and will be delivered before the deadline.

Non-Technical Users -

- **University Students** - For their precise math calculations where they must compute the power where x and y are really huge integers, students can utilise this scientific calculator.
- **Professors and Teachers** - They can use this tool for helping them in the projects and assignments.
- **Academic Researchers** - This scientific calculator is available for use by researchers. They can use it to assess how well different calculators perform in terms of accuracy, productivity, and turnaround time.

PROBLEM 2

2 Requirements Specification

2.1 List of Abbrevations

Table 1: Abbreviations.

Terms	Definition
FR	Funtional Requirement
NFR	Non-Functional Requirement

2.2 Constraints and Assumptions

- There are no default values; the end user must add the values for the base and exponent.
- Only real numbers should be entered by the user.
- A certain amount of base and exponent values can be entered by the user. Big Integers or decimals are not handled by the system.

3 Requirememts

3.1 Functional Requirements

-

ID	FR1
TYPE	Functional
VERSION	1.0
Priority	High
Risk	High
Description	Once the calculator is on the system asks the user to enter the value of x and y and values should be real number otherwise the system stops.
Rationale	In order to start the process we get the values from user.

-

ID	FR2
TYPE	Functional
VERSION	1.0
Priority	2
Risk	High
Description	When a user enters a value that is not a real number, the system should prompt the user to provide a real number and display an error message.
Rationale	Only real numbers should be used as input for calculations.

-

ID	FR3
TYPE	Functional
VERSION	1.0
Priority	2
Risk	Low
Description	The user can do the calculation get results then move on to next calculation with having him/her to re run the system or exiting the sytem.
Rationale	Easy Navigation and Usability of the System.

-

ID	FR4
TYPE	Functional
VERSION	1.0
Priority	2
Risk	Low
Description	The respective user of the system should be given a choice at the start of the program whether he or she wants to continue or exit.
Rationale	Only real numbers should be used as input for calculations.

-

ID	FR5
TYPE	Functional
VERSION	1.0
Priority	13
Risk	Low
Description	If the value entered of the base is 0 then the system should display an error message.
Rationale	Because 0 raised to the power anything is 0.

ID	FR6
TYPE	Functional
VERSION	1.0
Priority	4
Risk	Low
Description	Base is 0 and exponent is negative the system should display error message
Rationale	0 raised to the power of negative number is undefined.

ID	FR7
TYPE	Functional
VERSION	1.0
Priority	4
Risk	Low
Description	When base is negative and the power is fraction the result is undefined. So the system should display error message.
Rationale	Negative base cannot have fraction power as it is by rule undefined.

3.2 Non-Functional Requirements

ID	NFR1
TYPE	Non-Functional
VERSION	1.0
Priority	4
Risk	Low
Description	The error message displayed by the system should be relevant so that the user can understand the cause
Rationale	The type of mistake should be known to user.

ID	NFR2
TYPE	Non-Functional
VERSION	1.0
Priority	4
Risk	Low
Description	Easy Navigation for Interface.
Rationale	Usability of the system should be high .

ID	NFR3
TYPE	Non-Functional
VERSION	1.0
Priority	4
Risk	Low
Description	The result displayed should be of high accuracy and precision
Rationale	Correctness of the system is in question.

PROBLEM 3

4 Algorithm and its Subordinate Functions

4.1 Method-1

Algorithm 1: Power-Function (x,y)

Begin:
1. checkIfInputsAreValidNumbers(x,y)
2. IF x is 0 AND y $\leq$ 0 THEN
3. THROW EXCEPTION
4. ELSE IF x $\leq$ 0 AND y $>$ 0 and $<$1 THEN
5. THROW EXCEPTION
6. ELSE IF x and y are integers
7. SET finalResult to 1
8. SET COUNTER to 0
9. WHILE COUNTER $\leq$ y
10. UPDATE finalResult = finalResult * x
11. INCREMENT COUNTER by 1
12. PRINT finalResult
13.ELSE IF y is fraction
14. decimal_power(x,y)
22.PRINT finalResult

End

Procedure checkIfInputsAreValidNumbers(x,y)

Begin:
1. IF x $\in -\infty$ to $+\infty$ and y $\in -\infty$ to $+\infty$
2. SEND TRUE
3. ELSE
4. RAISE EXCEPTION and SEND FALSE

End

Procedure whole_answer_from_decimal_power(x,y)

```
Begin:
1. SET finalResult to 1
2. SET COUNTER TO Y
3. WHILE COUNTER>0
4.       finalResult = finalResult * x
5.       DECREMENT COUNTER by 1
6.       DECREMENT y by 1
7.        IF y<0
8.        BREAK LOOP
9. RETURN finalResult and y
End
```

Procedure power_greater_than_1(x,y)

```
Begin:
1. IF y>1
2.       whole_answer_from_decimal_power(x,y)
3. RETURN [finalResult and y]
End
```

Procedure Approx_Root_Value(x,y,Initial_Root$_Value, precision$)

```
Begin:
1. SET Initial Root_Value+= precision
2. [val,y]=whole_answer_from_decimal_power(x,y)
3. WHILE val<x:
4.        SET Initial Root Value+= precision
2.       [val,y]=whole_answer_from_decimal_power(x,y)
9. RETURN Root_Value
End
```

Procedure power_greater_than_1(x,y)

Begin:
1. SET numerator = exponent;
2. SET denominator_root = 1;
3. while NOT ((numerator * denominator_root) % 1 == 0):
4. INCREMENT denominator_root BY 1;
5.decimal_to_fraction = numerator, denominator$_root$;
6. SET accurate = 1;
7. Root_Value = Approx_Root_Value(base, decimal$_to_fraction$[1], 0, $accurate$);
8. WHILE(base < whole_answer_from_decimal_power(Root_Value,
decimal_to_fraction[1])[0] accurate > precision)
9. COMPUTE Root_Value -= accurate;
10. COMPUTE accurate *= 0.1;
11. GET Root_Value = Approx_Root_Value(base, decimal_to_fraction[1], Root_Value,
accurate)
12. GET finalResults = numerator, denominator_root, Root_Value
13.RETURN finalResults
End

Procedure decimal_power(x,y)

Begin:
1.SET finalResult= 1
2.IF(exponent >= 1)
3. GET exponentialValue = power_greater_than_1(base,exponent)
4. SET finalResult = finalResult * exponentialValue[0]
5. SET exponent = exponentialValue[1]
6.IF(exponent > 0 exponent < 1)
7. GET results = power_less_than_1(base,exponent)
8. SET finalResult = finalResult * whole_answer_from_decimal_power(results[2],
results[0]*results[1])[0]
9.RETURN finalResult
End

4.2 Method-2

Algorithm 2: Compute_power (x,y)

Begin:
1. checkIfInputsAreValidNumbers(x,y)
2. IF x is 0 AND y $\leq$ 0 THEN
3. THROW EXCEPTION
4. ELSE IF x $\leq$ 0 AND y $>$ 0 and $<$1 THEN
5. THROW EXCEPTION
6. ELSE IF x and y are integers
7. SET finalResult to 1
8. SET COUNTER to 0
9. WHILE COUNTER $\leq$ y
10. UPDATE finalResult = finalResult * x
11. INCREMENT COUNTER by 1
12. PRINT finalResult
13.ELSE IF y is fraction
14. SET finalResult to 1
15. SET COUNTER to 0
16. Extract Numerator 17. WHILE COUNTER $\leq$ y
18. UPDATE finalResult = finalResult * x
19. INCREMENT COUNTER by 1
20. Store Result to finalResult
21. Take N-th root of finalResult with Denominator being N.
22. PRINT finalResult

End

Procedure checkIfInputsAreValidNumbers(x,y)

begin:
1. IF x $\in -\infty$ to $+\infty$ and y $\in -\infty$ to $+\infty$
2. SEND TRUE
3. ELSE
4. RAISE EXCEPTION and SEND FALSE

end

4.3 Explanation

The two Power Function algorithms stated above each have various benefits and drawbacks in comparison to one another. The other is based on the algorithm of finding the nth root, whereas the first is combination of multiple mathematical rules. In order to create a precise calculator, the sub-functions are employed to compute and validate various scenarios.

4.4 Method Selection Reasoning

Selection of Method 1 depends on multiple factors.

4.4.1

Benefits of Selected Method:

- **Understandability:** Method 1 is highly modular and also has low coupling and high cohesion. It is easy to read and highly maintainable.
- **Exception Handling:** For all the error inputs exception handling has been done.
- **Usability:** It is very easy for a user to use this system as in-depth navigation path assistance is provided through proper instructions on the screen.

4.4.2

Disadvantages of Selected Method:

- **Code length:** Method 1 is big in comparison to method 2.
- **Complexity:** Method 1 has more space and time complexity than method 2.
- **Interface:** No UI is present the program runs on the console.

Method 2 Insights.

4.4.3

Benefits of Method 2:

- **Complexity:** Method 2 has less time and space complexity in comparison to method 1.
- **Readability:** Method 2 is small and concise hence easy to read..
- **High Cohesion:** Method 2 function are not dependent on other methods.

4.4.4

Disadvantages of Method 2:

- **Accuracy of results:** Method 2 results are less accurate in comparison to Method 1.
- **Exception Handling:** Not all the corner cases has been handled in this method.
- **Covering of problem domain:** It is not covering all the problem domain hence it does not work for every input.

5 Problem 4

5.1 Coding Files

1. F7.java - This file contains the main implementation of power function.
2. Test_cases.java - This file contains all the test cases relevant to the main file implementation.

5.2 DEBUGGER

A debugger is a software tool that can aid in the development of software by spotting code problems at different stages of the creation of an operating system or an application. Debuggers examine a test run to determine which lines of code were skipped.During the project's coding phase, Intellij was the IDE employed. Additionally, Intellij's built-in debugger was used to troubleshoot issues that arose during the coding phase.

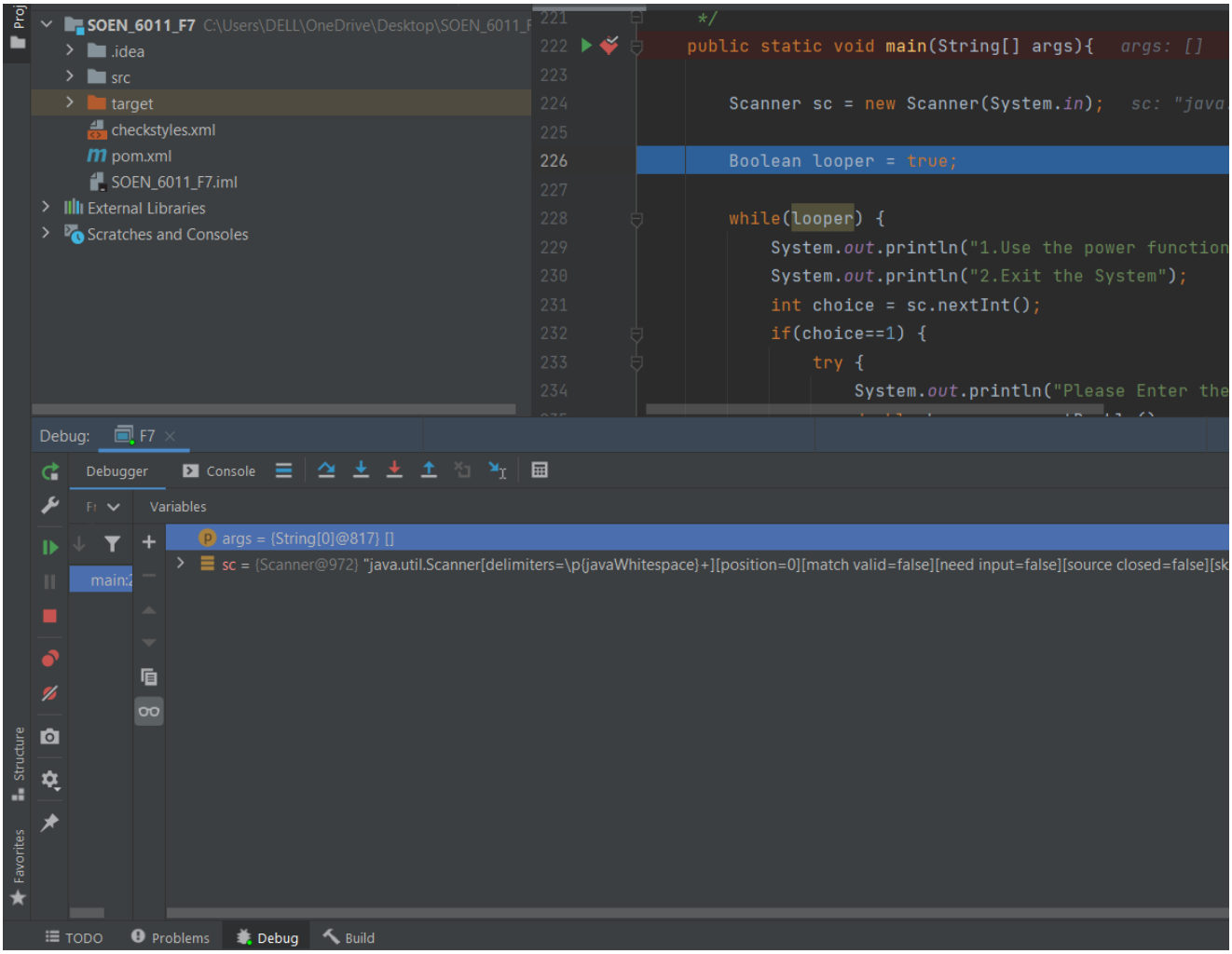
5.2.1 Advantages of Debugger

- When you debug your code, the IDE shows you variable values in the source code next to their usages and lets you change the values
- One can also use the Smart Step-Into action to choose the specific method call you want to debug.
- IntelliJ IDEA's built-in terminal lets you work with the command-line shell from your IDE without having to switch to a dedicated terminal application.
- The debugger interface has high usability and easy navigation.

5.2.2 Disadvantages of Debugger

- Slow performance or hangupus when starting the debugger.
- Too many features on the debuggers interface can confuse the beginner.
- Navigation can also be complex for the first time user.
- Because of the JVM's design, method breakpoints create a significant slowdown in the debugger because they are expensive and time consuming to test.

5.2.3 SNAPSHOT



5.3 Quality Attributes

5.3.1 Space-Efficient:

A measurement of how much memory is required for an algorithm to run is called space efficiency. In our project efficient data structures and algorithms has been used so that the overall requirement of the memory is less.

5.3.2 Portability:

A computer program is said to be portable if it can run on an operating system other than the one it was designed for without requiring extensive rewriting. In our case Java programming language has been used, its runs on a compiler and follows object oriented paradigm these qualities make it highly portable.

5.3.3 Maintainability:

The needs to be readable, testable, and reusable that's what makes it maintainable. Our code is highly modular, there is low coupling and high cohesion.

5.3.4 Correctness:

According to the standpoint of software engineering, correctness is the observance of the standards that describe how users can interact with the software and how the software should perform when used correctly. Our code conforms to the above mentioned standards.

5.3.5 Robustness:

When a piece of software can handle unexpected or wrong input, it is said to be robust. Our code is handling almost all the wrong cases that can happen in the case of power function.

5.3.6 Time-efficient:

A general word used to describe the dependability, speed, and programming style utilized in producing codes for an application is "time efficiency." In our project specific standard of coding style is followed in order to make it time efficient.

5.3.7 Usability:

The ability to modify or fix the code with ease and how reusable it is. It is practiced in our case to not add a lot of functionality to a single function.

5.4 Checkstyle

A static code analysis tool called Checkstyle is used in software development to determine whether Java source code complies with predetermined coding standards. It is therefore perfect

for initiatives that aim to impose a coding standard. It can be made to support practically any coding standard and is highly flexible.

5.4.1 Advantages of Checkstyle

- The capacity to make your own rules. A sizable number of styles are defined by Eclipse or Intellij, but Checkstyle offers more, and you can add your own unique rules.
- Improved outside tooling Due to the fact that Checkstyle was initially intended to be a standalone framework, it is much simpler to combine it with your external tools.
- Portable Between IDE's.

5.4.2 SNAPSHOT

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>groupId</groupId>
    <artifactId>CosInverse</artifactId>
    <version>1.0-SNAPSHOT</version>
    <build>
        <plugins>
            <plugin>
                <!--https://maven.apache.org/plugins/maven-checkstyle-plugin/-->
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-checkstyle-plugin</artifactId>
                <version>3.1.2</version>
                <configuration>
                    <configLocation>checkstyle-checker.xml</configLocation>
                    <encoding>UTF-8</encoding>
                    <consoleOutput>true</consoleOutput>
                    <failsOnError>true</failsOnError>
                </configuration>
            </plugin></plugins>
    </build>
    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>RELEASE</version>
            <scope>test</scope>
        </dependency>
```

5.4.3 Disadvantages of Checkstyle

- It is impossible to ascertain the complete inheritance hierarchy of type.
- There is no access to the contents of other files because files are processed one at a time.
- If the code cannot be compiled using the javac compiler, the parsing errors are quite challenging to understand.

6 Problem-5

6.1 Junit

A unit testing framework for the Java programming language is called JUnit. It is a family of unit testing frameworks known as xUnit, and it is a key component in test-driven development.

In order to build up the test data for a piece of code that may be tested first and then implemented, JUnit promotes the principle of "first testing then writing." Similar to "test a little, code a little, test a little, code a little," is this strategy. Programmer stress and debugging time are decreased as a result of the increased productivity of the programmer and the stability of the programme code. cases that identify and satisfy the functional requirement are traced below.

6.1.1 Test Case 1

ID $\rightarrow$ $test\_1()$
Function $\rightarrow$ $F7.power\_positive(double, double)$
Input $\rightarrow$ $(1.0, 5.0)$
Output $\rightarrow$ 1.0
Expected $\rightarrow$ $true$
Result $\rightarrow$ $Pass$

6.1.2 Test Case 2

ID $\rightarrow$ $test\_2()$
Function $\rightarrow$ $F7.power\_negative(double, double)$
Input $\rightarrow$ $(2.0, -4.0)$
Output $\rightarrow$ 0.25
Expected $\rightarrow$ $True$
Result $\rightarrow$ $Pass$

6.1.3 Test Case 3

ID $\rightarrow$ $test\_3()$
Function $\rightarrow$ $F7.Controller(2, -2.6)(double, double)$
Input $\rightarrow$ $(2.0, -2.6)$
Output $\rightarrow$ 0.164938
Expected $\rightarrow$ $True$
Result $\rightarrow$ $Pass$

6.1.4 Test Case 4

ID $\rightarrow$ $test\_4()$
Function $\rightarrow$ $F7.Controller(double, double)$
Input $\rightarrow$ $(2.0, 3.7)$
Output $\rightarrow$ 12.996038
Expected $\rightarrow$ $True$
Result $\rightarrow$ $Pass$

6.1.5 Test Case 5

ID $\rightarrow$ $test\_5()$
Function $\rightarrow$ $F7.power\_positive(double, double)$
Input $\rightarrow$ $(1.0, 5.0)$
Output $\rightarrow$ 2.0
Expected $\rightarrow$ $False$
Result $\rightarrow$ $Pass$

6.1.6 Test Case 6

ID $\rightarrow$ $test\_6()$
Function $\rightarrow$ $F7.decimal\_power(double, double)$
Input $\rightarrow$ $(1.0, -5.0)$
Output $\rightarrow$ 15.0
Expected $\rightarrow$ $False$
Result $\rightarrow$ $Pass$

6.1.7 Test Case 7

ID $\rightarrow$ $test\_7()$
Function $\rightarrow$ $F7.decimal\_power(double, double)$
Input $\rightarrow$ $(2.0, -2.6)$
Output $\rightarrow$ 6.050000
Expected $\rightarrow$ $False$
Result $\rightarrow$ $Pass$

6.1.8 Test Case 8

ID $\rightarrow$ $test\_8()$
Function $\rightarrow$ $F7.decimal\_power(double, double)$
Input $\rightarrow$ $(2.0, 3.7)$
Output $\rightarrow$ 12.900000
Expected $\rightarrow$ $False$
Result $\rightarrow$ $Pass$

6.1.9 Test Case 9

ID $\rightarrow$ $test\_9()$
Function $\rightarrow$ $F7.Controller(double, double)$
Input $\rightarrow$ $(0, 0)$
Output $\rightarrow$ -1.0
Expected $\rightarrow$ $True$
Result $\rightarrow$ $Pass$

6.1.10 Test Case 10

ID – test_10()
Function — F7.decimal_power(double, double)
Input— (0.0,3.7)
Output $\rightarrow$ 0
Expected — True
Result — Pass

References

[1] Junit,
https://www.tutorialspoint.com/junit/junit$_o verview.html$

[2] Power Function,
https://www.storyofmathematics.com/power-function

[3] Power Function Algebraic Expression,
https://study.com/academy/lesson/what-is-a-power-function-equations-and-graphs.html

[4] Checkstyle,
https://plugins.jetbrains.com/plugin/1065-checkstyle-idea

[5] Code Hint,
https://www.geeksforgeeks.org/write-a-c-program-to-calculate-powxn/