Error handling in JAVA

Subject: CSW2(CSE3141) **Section: All**

Session: Feb 2024 to April 2024 **Branch: CSE&CSIT**

……

1. Implement a Java program to handle **NullPointerException**.
2. You are given a string containing alpha-numeric characters. Design a Java program that displays the numeric characters if it is preceded by a vowel and consonant in the given string. If such numeric characters are not present in the given string, display appropriate message. If the input string is null or empty, throw a **NullPointerException** with an appropriate error message. Ensure that the program handles any potential exceptions gracefully.
3. Implement a custom **NullPointerException** class named **CustomNullPointerException** that mimics the behavior of the standard **NullPointerException**, but instead of using default error messages or null references, it takes a String message as its constructor argument. Your task is to create this custom exception class and demonstrate its usage in a Java program.
4. Implement a Java program that reads a file path from the command line argument and attempts to read the contents of the file. If the file path is null or points to a non-existent file, throw a custom **FileNotFoundException**. If the file exists but cannot be read due to permission issues, throw a custom **FileReadPermissionException**. Your task is to create these custom exception classes and handle them appropriately in your program.
5. Develop a program that performs complex mathematical (may have log, trigonometric and algebraic functions) computations. Handle unchecked **NullPointerException** gracefully using **try-catch** block and provide a meaningful error message.
6. Write a Java program to handle **NumberFormatException**.
7. Create a method that takes a string input and converts it to an integer. Handle **NumberFormatException** using try-catch block and prompt the user to enter a valid number upon exception.
8. Write a Java program to find the quare root of integer numbers. Demonstrate the use of **try-catch** block to handle **ArithmeticException.**
9. Create a program that divides two numbers input by the user. Handle the possibility of division by zero using try-catch block to catch **ArithmeticException**.
10. Implement a Java program that calculates the value of the expression (sin(x) + cos(x)) / tan(x) for a given value of x. Handle scenarios where **x is close to multiples of π/2** to avoid division by zero errors.
11. Write a Java program that computes the value of the function log(sin(x) + cos(x)) / (tan(x) - cot(x)) for a given value of x. Ensure proper handling of exceptions that **may occur due to invalid mathematical operations**.
12. Create a Java application that calculates the value of the expression sqrt(abs(sin(x) * cos(x))) / (tan(x) + 1) for a given value of x. Handle cases **where x leads to division by zero or negative values** inside the square root function.

13. Design a Java program that evaluates the value of the function (sin(x) * cos(x)) / (sin(x) + cos(x)) for a given value of x. Handle potential arithmetic exceptions that **may arise due to invalid mathematical operations**.
14. Implement a Java application that computes the value of the expression log(abs(sin(x) + cos(x))) / (tan(x) - cot(x)) for a given value of x. Ensure proper error handling for potential arithmetic exceptions and **negative values inside the logarithmic function**.
15. Demonstration of use nested try-catch block. Write a Java program to handle **NumberFormatException** in outer **try-catch** block and ArithmeticException inside the inner **try-catch** block**.**
16. Create a Java program to handle **ArrayIndexOutOfBoundsException.**
17. Implement a Java program that involves dynamic data structures such as linked lists or trees, where elements are stored in arrays. Introduce scenarios, where accessing elements beyond the bounds of the array backing the data structure results in **ArrayIndexOutOfBoundsException**. Your task is to implement bounds checking and handle these exceptions effectively while maintaining the integrity of the data structure.
18. Develop a recursive algorithm implemented in Java that traverses or manipulates arrays. Introduce scenarios where the recursion reaches beyond the bounds of the array, resulting in **ArrayIndexOutOfBoundsException**. Your task is to handle these exceptions within the recursive algorithm and ensure proper termination of recursion.
19. Implement a Java program that performs complex manipulations on an array of integers. The program should involve operations such as sorting, searching, and accessing elements at various indices. Introduce scenarios, where accessing elements beyond the bounds of the array leads to **ArrayIndexOutOfBoundsException**. Your task is to handle these exceptions gracefully and ensure the program continues execution without crashing.
20. Develop a recursive algorithm implemented in Java that traverses or manipulates arrays. Introduce scenarios where the recursion reaches beyond the bounds of the array, resulting in **ArrayIndexOutOfBoundsException**. Your task is to handle these exceptions within the recursive algorithm and ensure proper termination of recursion.
21. Design a Java program that performs matrix operations such as addition, multiplication, and transpose. Introduce scenarios, where accessing elements beyond the bounds of the matrix results in **ArrayIndexOutOfBoundsException**. Your task is to handle these exceptions effectively and provide meaningful error messages indicating the nature of the exception.
22. Create a custom checked exception class named **CustomCheckedException**. Use this exception in your program to handle a specific error condition and demonstrate its usage using try-catch block.
23. Write a program that reads data from a file and performs some processing. Handle checked **IOException** by using try-catch block to catch and handle the exception.
24. Write a Java program to demonstrate a checked exception (e.g., **FileNotFoundException**) being thrown and caught using try-catch block.
25. Implement a method that reads an integer from the user but handles **InputMismatchException** using try-catch block.
26. Implement **try-catch-finally** blocks to handle **ClassNotFoundException** and **MethodNotFoundException**.
27. Write a program to handle **ClassCastException**.

28. Implement a Java program to handle **StackOverflowError.**