Ex.No. 01	To generate Electricity Bill

Aim:

To develop a Java application is to generate electricity bills for consumers based on their electricity consumption and connection type. The program should calculate the bill amount using the provided tariff rates and display the consumer details along with the calculated bill amount.

Algorithm:

1. Start the program.
2. Create a Scanner object to read user input.
3. Prompt the user to enter the consumer details (consumerNo, consumerName, prevMonthReading, currMonthReading, and ebConnectionType).
4. Read the consumer details from the user.
5. Create an instance of the EBBill class and name it ebBill.
6. Check if ebConnectionType is equal to "Domestic" (case-insensitive).
    a. If true, create a new DomesticBill object and assign it to ebBill, passing the consumer details to the constructor.
    b. If false, create a new CommercialBill object and assign it to ebBill, passing the consumer details to the constructor.
7. Call the displayBill() method on ebBill to display the bill details.
8. End the program.

Ex.No. 02	kth smallest element in an unsorted array

Aim:

To write a Java program to find the kth smallest element in an unsorted array.Create two separate classes, Main for the main program and another K^{th}SmallestElement for finding the kth smallest element.

Algorithm:

1. Start the program.
2. Create a Scanner object to read user input.
3. Read the value of size from the user.
4. Create an integer array arr of size size.
5. Iterate i from 0 to size - 1:
    a. Read the value from the user and store it in arr[i].
6. Read the value of k from the user.
7. Create an instance of the KthSmallestElement class and name it kthSmallest.
8. Call the findKthSmallest() method on kthSmallest, passing arr, 0, size - 1, and k as arguments, and store the result in the variable kth.
9. Print the value of kth.
10. End the program.

Ex.No. 03	Printing Sub Array with given sum

Aim:

To write a Java program to find the sub-array with the given sum.

Algorithm:

1. Start the program.
2. Create a Scanner object to read user input.
3. Read the value of n from the user, representing the size of the input array.
4. Create an integer array arr of size n.
5. Iterate i from 0 to n - 1:
    a. Read the value from the user and store it in arr[i].
6. Read the value of targetSum from the user.
7. Create a boolean variable foundSubarray and set it to false.
8. Iterate i from 0 to n - 1:
    a. Create an integer variable currSum and set it to 0.
    b. Iterate j from i to n - 1:
        i. Add arr[j] to currSum.
        ii. Check if currSum is equal to targetSum.
            1. If true, print "Subarray found: [" on the console.
            2. Iterate k from i to j:
                a. Print arr[k] followed by a space on the console.
        iii. Print "]" on the console.
        iv. Set foundSubarray to true.
        v. Break out of the inner loop.
    c. Check if foundSubarray is true.
        i. If true, break out of the outer loop.
9. Check if foundSubarray is false.
    a. If true, print "Subarray not found" on the console.
10. End the program.

Ex.No. 04	Remove duplicate elements in an array

Aim:

To write a Java program to remove duplicate elements in an array.

Algorithm:

1. Start the program.
2. Create a Scanner object to read user input.
3. Read the value of n from the user, representing the size of the input array.
4. Create an integer array arr of size n.
5. Iterate i from 0 to n - 1:
    a. Read the value from the user and store it in arr[i].
6. Create an instance of the DuplicateRemover class, passing the arr array as a parameter.
7. Invoke the removeDuplicates method on the DuplicateRemover instance and store the returned array in uniqueArr.
8. Print "Array with duplicates removed: [" on the console.
9. Iterate i from 0 to the length of uniqueArr - 1:
    a. Print uniqueArr[i] followed by a space on the console.
10. Print "]" on the console.
11. End the program.

Ex.No. 05	N^{th} digit in the integer sequence

Aim:

To write a Java program to accept an integer value N and print the N^{th} digit in the integer sequence.

Algorithm:

1. Start the program.
2. Create a Scanner object to read user input.
3. Read the value of n from the user.
4. Initialize a variable count to 0 to keep track of the number of digits encountered.
5. Initialize a variable number to 1 to represent the current number being considered.
6. Initialize a variable lastNumber to 0 to store the last number encountered.
7. Enter a loop while count is less than n:
    a. Set lastNumber to the value of number.
    b. Increment number by 1.
    c. Add the number of digits in lastNumber to count by invoking the countDigits method.
8. Calculate the difference diff between count and n.
9. Check if diff is equal to 0:
    a. If true, the nth digit is the last digit of lastNumber, so output lastNumber % 10.
    b. If false, proceed to the next step.
10. Set a temporary variable temp to the value of lastNumber.
11. Enter a loop while diff is greater than 0:
    a. Divide temp by 10 to remove the last digit.
    b. Decrement diff by 1.
12. Output temp % 10, which represents the nth digit of the sequence.
13. End the program.

Ex.No. 06	String operations using ArrayList

Aim:

To write a Java program to perform string operations using ArrayList and write functions append, insert, search, and list all strings starting with the given letter.

Algorithm:

1. Start the program.
2. Create an ArrayList object named list to store strings.
3. Create a Scanner object named scanner to read user input.
4. Enter an infinite loop using while (true) to continuously prompt for user input.
5. Read the value of choice from the user.
6. Use a switch statement based on the value of choice to perform different actions:
    a. Case 1:
        i. Read a string str from the user.
        ii. Invoke the append method, passing list and str as arguments.
        iii. Break from the switch statement.
    b. Case 2:
        i. Read a string str from the user.
        ii. Read an integer index from the user.
        iii. Invoke the insert method, passing list, str, and index as arguments.
        iv. Break from the switch statement.
    c. Case 3:
        i. Read a string str from the user.
        ii. Invoke the search method, passing list and str as arguments.
        iii. Break from the switch statement.
    d. Case 4:
        i. Read a character letter from the user.
        ii. Invoke the listStrings method, passing list and letter as arguments.
        iii. Break from the switch statement.
    e. Case 5:
        i. Close the scanner object.
        ii. Output "Exiting."
        iii. Terminate the program using System.exit(0).
7. End the loop.
8. Define the append method, which takes an ArrayList<String> object list and a string str as parameters:
    a. Add the string str to list using the add method.
    b. Output str + " appended.".
9. Define the insert method, which takes an ArrayList<String> object list, a string str, and an integer index as parameters:
    a. Insert the string str at the specified index in list using the add method.
    b. Output str + " inserted at index " + index + ".".
10. Define the search method, which takes an ArrayList<String> object list and a string str as parameters:

    a. Check if list contains the string str using the contains method.
    b. If str is found in list, output str + " found.".
    c. If str is not found in list, output str + " not found.".
11. Define the listStrings method, which takes an ArrayList<String> object list and a character letter as parameters:
    a. Iterate over each string str in list.
    b. If the first character of str is equal to letter, output str.
12. End the program.

Ex.No. 07	Frequency of words in a given text

Aim:

To write a Java program to find the frequency of words in each text.

Algorithm:

1. Start the program.
2. Create a Scanner object named scanner to read user input.
3. Read a line of text from the user and store it in a string variable named text. Convert the text to lowercase.
4. Split the text into words using the split("\\W+") method and store the words in a string array named words. This splits the text using any non-word characters as delimiters.
5. Create a LinkedHashMap named wordFrequencyMap to store word frequencies while maintaining the order.
6. Iterate over each word word in words:
    a. If wordFrequencyMap contains the word as a key, increment its corresponding value by 1.
    b. If wordFrequencyMap does not contain the word as a key, add it to the map with a value of 1.
7. Iterate over each word word in words:
    a. If wordFrequencyMap contains the word as a key:
        i. Output word followed by a colon and its corresponding value from wordFrequencyMap.
        ii. Remove the word from wordFrequencyMap.
8. End the program.

Ex.No. 08	Number pattern

Aim:

To write a Java program to find the number of patterns.

Algorithm:

1. Start the program.
2. Create a Scanner object named sc to read user input.
3. Read a string from the user and store it in a variable named str.
4. Call the function patternCount with the input string str as an argument.
5. Print the return value of patternCount as the output.
6. End the program.

Function patternCount:

1. Start the function with a parameter str, which represents the input string.
2. Initialize a variable last with the first character of str.
3. Initialize variables i and counter to 1 and 0, respectively.
4. Enter a while loop with the condition i <str.length():
    a. If the current character at index i in str is '0' and the last character last is '1', enter a nested while loop.
        i. Increment i if the current character at index i is '0'.
        ii. Check if i is equal to the length of str. If true, break out of the loop.
        iii. Check if the current character at index i in str is '1'. If true, increment the counter by 1.
    b. Check if i is equal to the length of str. If true, break out of the loop.
    c. Update the last character with the current character at index i in str.
    d. Increment i by 1.
5. Return the value of counter.
6. End the function.

Ex.No. 09	Longest repeating sequence of string

Aim:

To write a Java program to find the longest repeating sequence in a string.

Algorithm:

1. Start the program.
2. Create a Scanner object named scanner to read user input.
3. Read a string from the user and store it in a variable named inputString.
4. Call the function findLongestRepeatingSequence with the input string inputString as an argument and store the result in a variable named longestRepeatingSequence.
5. If longestRepeatingSequence is empty, print "No repeating sequence found".
6. Otherwise, print longestRepeatingSequence.
7. End the program.

Function findLongestRepeatingSequence:

1. Start the function with a parameter inputString, which represents the input string.
2. Initialize a variable inputStringLength with the length of inputString.
3. Initialize an empty string variable longestRepeatingSequence.
4. Iterate from index i = 0 to inputStringLength - 1 using a for loop:
    a. Iterate from index j = i + 1 to inputStringLength - 1 using another for loop:
        i. Create a substring substring from inputString starting at index i and ending at index j.
        ii. Initialize a variable substringLength with the length of substring.
        iii. Iterate from index k = j to inputStringLength - 1 using another for loop:
            1. Check if k + substringLength is less than or equal to inputStringLength.
                a. If true, check if the substring from inputString starting at index k and ending at index k + substringLength is equal to substring.
                    i. If true, compare the length of substring with the length of longestRepeatingSequence.
                        1. If substring is longer than longestRepeatingSequence, update longestRepeatingSequence with substring.
5. Return the value of longestRepeatingSequence.
6. End the function.

Ex.No. 10	Reverse a set of words and count the frequency

Aim:

To write a Java program to reverse a set of words and count the frequency of each letter in the string.

Algorithm:

1. Start the program.
2. Create a Scanner object named scanner to read user input.
3. Read a line of text from the user and store it in a variable named inputString. Convert the string to lowercase.
4. Split the inputString into an array of words using the regular expression "\W+" and store it in a variable named words.
5. Create a StringBuilder object named reversedString to build the reversed string.
6. Iterate backwards through the words array from index words.length - 1 to 0 using a for loop:
    a. Append each word followed by a space to the reversedString object.
7. Convert the reversedString object to a string using the toString() method and trim any leading or trailing spaces. Store the result in a variable named outputString.
8. Create a HashMap named letterFrequencyMap to store the frequency of each letter.
9. Iterate through each character c in the inputString using a for loop:
    a. Check if c is a letter using the Character.isLetter() method.
        i. If true, check if letterFrequencyMap contains c as a key.
            1. If true, increment the value associated with key c by 1 in letterFrequencyMap.
            2. If false, add a new entry in letterFrequencyMap with key c and value 1.
10. Print the "Reversed string: " followed by the value of outputString.
11. Print "Letter frequencies:".
12. Iterate through each entry in letterFrequencyMap using a for-each loop:
    a. Print the key of the entry followed by ": " and then the value of the entry.
13. End the program.

Ex.No. 11	Java interface for ADT Stack

Aim:

To design a Java interface for ADT Stack, implement this interface using an array and built-in classes and provide necessary exception handling in both implementations.

Algorithm:

1. Start the program.
2. Create a Scanner object named scanner to read user input.
3. Read an integer from the user and store it in a variable named maxSize. This represents the maximum size of the stack.
4. Create a stack object of type ArrayStack<Integer> named stack with the specified maxSize.
5. Enter an infinite loop using while (true) to display the menu and handle user choices:
    a. Read an integer from the user and store it in a variable named choice.
    b. Use a switch statement based on choice to handle different menu options.
    c. For each case:
        i. Case 1: Read an integer element from the user and attempt to push it onto the stack using stack.push(element).
            1. If successful, print a success message.
            2. If a StackOverflowException is caught, print an error message.
        ii. Case 2: Attempt to pop an element from the stack using stack.pop().
            1. If successful, print the popped element.
            2. If a StackUnderflowException is caught, print an error message.
        iii. Case 3: Attempt to peek at the top element of the stack using stack.peek().
            1. If successful, print the peeked element.
            2. If a StackUnderflowException is caught, print an error message.
        iv. Case 4: Get the size of the stack using stack.size() and print it.
        v. Case 5: Check if the stack is empty using stack.isEmpty() and print the corresponding message.
        vi. Case 6: Check if the stack is full using stack.isFull() and print the corresponding message.
        vii. Case 7: Print an exit message and terminate the program using System.exit(0).
        viii. Default: Print an invalid choice message.
    d. Print a new line to separate the outputs.
6. End the program.

Ex.No. 12	Remove all the occurrences of string S2 in string S1

Aim:

To write a Java program to remove all the occurrences of string S2 in string S1 and print the remaining.

Algorithm:

1. Start the program.
2. Create a Scanner object named scanner to read user input.
3. Read a string from the user and store it in a variable named s1. This represents the original string.
4. Read another string from the user and store it in a variable named s2. This represents the string to be removed from s1.
5. Create a StringBuilder object named sb to store the resulting string after removing s2.
6. Get the length of s2 and store it in a variable named s2Length.
7. Iterate over each character in s1 using a for loop with index i ranging from 0 to the length of s1 - 1:
    a. Check if i is less than or equal to the difference between the length of s1 and s2Length and if the substring of s1 from i to i + s2Length is equal to s2.
        i. If true, skip the next s2Length characters by incrementing i by s2Length - 1.
        ii. This effectively skips over the occurrences of s2 in s1.
    b. If false, append the character at index i from s1 to the StringBuilder sb.
8. Convert the StringBuilder sb to a string and store it in a variable named result.
9. Print the resulting string result.
10. End the program.

Ex.No. 13	Copy the content of one file to another

Aim:

To write a Java program to read and copy the content of one file to another by handling all file-related exceptions.

Algorithm:

1. Start the program.
2. Create a Scanner object named scanner to read user input.
3. Read a string from the user and store it in a variable named filename. This represents the name of the file to write.
4. Try to open the file for writing using a BufferedWriter and FileWriter inside a try-with-resources block:
    a. If successful:
        i. Enter a loop to read lines of input from the user until an empty line is encountered:
            1. Read a line of input from the user and store it in a variable named line.
            2. Write the line to the file using the writer.write() method.
            3. Write a newline character to the file using the writer.newLine() method.
        ii. Print "Contents written to file successfully!" to indicate successful writing.
    b. If an IOException occurs, print "Error writing to file: " followed by the exception message, and return.
5. Read another string from the user and store it in a variable named copyFilename. This represents the name of the file to copy to.
6. Try to open the original file for reading and the copy file for writing using BufferedReader and BufferedWriter inside separate try-with-resources blocks:
    a. If successful:
        i. Enter a loop to read lines from the original file until the end of the file is reached:
            1. Read a line from the original file and store it in a variable named line.
            2. Write the line to the copy file using the writer.write() method.
            3. Write a newline character to the copy file using the writer.newLine() method.
        ii. Print "File contents copied successfully!" to indicate successful copying.
    b. If an IOException occurs, print "Error copying file: " followed by the exception message, and return.
7. Read another string from the user and store it in a variable named displayFilename. This represents the name of the file to display.

8. Try to open the file for reading using a BufferedReader inside a try-with-resources block:
    a. If successful:
        i. Enter a loop to read lines from the file until the end of the file is reached:
            1. Read a line from the file and store it in a variable named line.
            2. Print line to display the contents of the file.
    b. If an IOException occurs, print "Error reading file: " followed by the exception message.
9. End the program.

Ex.No. 14	Print the number of unique string values

Aim:

To write a Java program to print the number of unique string values that can be formed by rearranging the letters in the string S.

Algorithm:

1. Start the program.
2. Create a Scanner object named scanner to read user input.
3. Read a string from the user and store it in a variable named s. This represents the input string.
4. Create an empty Set named uniqueStrings to store unique string values.
5. Call the permute function, passing s, 0 as the starting index (l), s.length() - 1 as the ending index (r), and uniqueStrings as arguments.
6. Print the number of unique string values in uniqueStrings using the size() method.
7. End the program.

Function permute:

1. Check if l is equal to r:
    a. If true, it means all characters have been fixed, so add the string s to the set.
    b. If false, proceed to the next step.
2. Enter a loop from i = l to i <= r:
    a. Call the swap function, passing s, l, and i as arguments, and assign the result back to s.
    b. Recursively call the permute function, passing s, l + 1, r, and set as arguments.
    c. Call the swap function again, passing s, l, and i as arguments, and assign the result back to s.
3. End the function.

Function swap:

1. Convert the string s to a character array named charArray.
2. Swap the characters at indices i and j in charArray.

3. Convert charArray back to a string and return the result.

Ex.No. 15	Abstract class

Aim:

To write a Java Program to create an abstract class named Shape that contains two integers and an empty method named printArea().

Algorithm:

1. Start the program.
2. Create a Scanner object named sc to read user input.
3. Create an instance of the Rectangle class named rect.
4. Read an integer from the user and assign it to rect.dimension1.
5. Read another integer from the user and assign it to rect.dimension2.
6. Call the printArea method of rect to calculate and print the area of the rectangle.
7. Call the numberOfSides method of rect to print the number of sides of the rectangle.
8. Create an instance of the Triangle class named tri.
9. Read an integer from the user and assign it to tri.dimension1.
10. Read another integer from the user and assign it to tri.dimension2.
11. Call the printArea method of tri to calculate and print the area of the triangle.
12. Call the numberOfSides method of tri to print the number of sides of the triangle.
13. Create an instance of the Circle class named cir.
14. Read an integer from the user and assign it to cir.dimension1.
15. Call the printArea method of cir to calculate and print the area of the circle.
16. Call the numberOfSides method of cir to print the number of sides of the circle.
17. Close the scanner (sc).
18. End the program.

Ex.No. 16	Matrix manipulations

Aim:

To write a Java program to perform Matrix manipulations – Addition, Subtraction, and Multiplication.

Algorithm:

1. Start the program.
2. Create a Scanner object named input to read user input.
3. Read the number of rows from the user and store it in the variable rows.
4. Read the number of columns from the user and store it in the variable columns.
5. Create two matrices matrix1 and matrix2 of size rows x columns.
6. Prompt the user to enter the elements of matrix1.
7. Read the elements of matrix1 from the user and store them in the corresponding positions.
8. Prompt the user to enter the elements of matrix2.
9. Read the elements of matrix2 from the user and store them in the corresponding positions.
10. Create a matrix sum of size rows x columns to store the sum of matrix1 and matrix2.
11. Compute the sum of corresponding elements from matrix1 and matrix2 and store it in sum.
12. Create a matrix diff of size rows x columns to store the difference of matrix1 and matrix2.
13. Compute the difference of corresponding elements from matrix1 and matrix2 and store it in diff.
14. Create a matrix product of size rows x columns to store the product of matrix1 and matrix2.
15. Compute the product of matrix1 and matrix2 using nested loops and store it in product.
16. Define a method displayMatrix that takes a matrix as input and displays its elements.
17. Print "Matrix 1:".
18. Call the displayMatrix method with matrix1 as the argument.
19. Print "Matrix 2:".
20. Call the displayMatrix method with matrix2 as the argument.
21. Print "Matrix Sum:".
22. Call the displayMatrix method with sum as the argument.
23. Print "Matrix Difference:".
24. Call the displayMatrix method with diff as the argument.
25. Print "Matrix Product:".
26. Call the displayMatrix method with product as the argument.
27. Close the input scanner.
28. End the program.

Ex.No. 17	JDBC

Aim:

To write a Java (JDBC) program to connect with the MySQL database and insert the given data in the table and display the table contents using ResultSet.

Algorithm:

1. Start the program.
2. Define the MySQL database URL, username, password, and the INSERT query to insert data into the STUDENT table.
3. Create a Scanner object named sc to read user input.
4. Load the MySQL JDBC driver using Class.forName.
5. Establish a database connection using DriverManager.getConnection with the specified URL, username, and password.
6. Create a PreparedStatement object st with the INSERT query.
7. Read the number of rows to insert from the user and store it in variable n.
8. Iterate n times using a for loop:
    a. Read the student ID from the user and store it in the variable id.
    b. Read the student name from the user and store it in the variable name.
    c. Read the average marks from the user and store it in the variable average_marks.
    d. Set the parameter values in the PreparedStatement object st using setInt and setString methods.
    e. Execute the update using st.executeUpdate() to insert the row into the STUDENT table.
9. Execute a SELECT query to retrieve all the rows from the STUDENT table using st.executeQuery and store the result in a ResultSet object rs.
10. Iterate over the rs using a while loop:
    a. Print the student ID, name, and average marks using rs.getInt, rs.getString, and rs.getInt methods respectively.
11. Close the PreparedStatement object st and the database connection con.
12. End the program.

Ex.No. 18	Pay slip calculation

Aim:

The aim of the given Java program is to generate a pay slip for different categories of employees (Programmer, Assistant Professor, Associate Professor, Professor) based on their basic pay.

Algorithm:

1. Start the program.
2. Create a Scanner object named sc to read user input.
3. Declare an integer variable choice to store the user's menu choice.
4. Enter a do-while loop that continues until the user chooses to exit (choice = 5).
5. Within the loop:
    a. Read the user's choice using sc.nextInt() and store it in the variable choice.
    b. Use a switch statement to handle different choices:
        i. Case 1: Programmer
            1. Read the employee details (name, ID, address, mail ID, mobile number, basic pay) from the user.
            2. Create a Programmer object p with the entered details.
            3. Call the generatePaySlip() method of p.
            4. Break from the switch statement.
        ii. Case 2: Assistant Professor
            1. Read the employee details from the user.
            2. Create an AssistantProfessor object ap with the entered details.
            3. Call the generatePaySlip() method of ap.
            4. Break from the switch statement.
        iii. Case 3: Associate Professor
            1. Read the employee details from the user.
            2. Create an AssociateProfessor object asp with the entered details.
            3. Call the generatePaySlip() method of asp.
            4. Break from the switch statement.
        iv. Case 4: Professor
            1. Read the employee details from the user.
            2. Create a Professor object pr with the entered details.
            3. Call the generatePaySlip() method of pr.
            4. Break from the switch statement.
        v. Case 5: Exit
            1. Print "Exiting..." message.
            2. Break from the switch statement.
        vi. Default: Invalid choice
            1. Print "Invalid choice!" message.
    c. Continue the loop until the user chooses to exit (choice = 5).
6. Close the Scanner object sc.
7. End the program.

Ex.No. 19	Exception Handling Implementation – Student Details

Aim:

Towrite a Java program is to handle exceptions for validating the age and name of a student while creating a student object. It checks if the age is between 15 and 21 and if the name contains only alphabets. If the age or name is not valid, custom exceptions are thrown and caught.

Algorithm:

1. Start the program.
2. Create custom exception classes AgeNotWithInRangeException and NameNotValidException.
3. Define the toString() method in each exception class to provide a custom error message when the exception is thrown.
4. Create a class Student with member variables roll, age, name, and course.
5. Define a default constructor Student() that initializes the member variables to default values.
6. Define a parameterized constructor Student(int r, String n, int a, String c) that accepts roll, name, age, and course as parameters.
7. Inside the parameterized constructor:
    a. Initialize the roll and course variables with the passed values.
    b. Initialize a variable l to store the length of the name and temp as 0.
    c. Iterate through each character in the name using a for loop.
    d. Check if the character is not an alphabet (i.e., it is not between 'A' and 'Z' or 'a' and 'z').
    e. If a non-alphabetic character is found, set temp to 1.
    f. Use a try-catch block to handle the NameNotValidException.
        i. If temp is 1, create an object of NameNotValidException and throw it.
        ii. If the exception is caught, print the exception message.
        iii. Otherwise, set the name to the passed value.
    g. Use another try-catch block to handle the AgeNotWithInRangeException.
        i. If the age is not between 15 and 21, create an object of AgeNotWithInRangeException and throw it.
        ii. If the exception is caught, print the exception message.
        iii. Otherwise, set the age to the passed value.
8. Define a method display() to display the student's details (roll, name, age, and course).
9. Create a class StudentDemo with the main method.
10. Create a Scanner object br to read user input.
11. Read the roll, name, age, and course from the user using br.nextLine() and Integer.parseInt(br.nextLine()).
12. Create a Student object s by invoking the parameterized constructor with the entered values.
13. Call the display() method of the s object to print the student's details.
14. Close the Scanner object br.
15. End the program.

	Exception Handling Implementation – Arithmetic operations

Ex.No. 20	

Aim:

To write a Java program is to perform arithmetic operations based on user input and handle possible exceptions like division by zero and invalid operators.

Algorithm:

1. Start the program.
2. Create a class named ArithmeticOperations.
3. Inside the main method:
    a. Create a Scanner object sc to read user input.
    b. Wrap the entire code inside a try-catch block to handle exceptions.
    c. Within the try block, perform the following steps:
        i. Read the first operand num1 using sc.nextInt().
        ii. Read the operator character using sc.next().charAt(0).
        iii. Read the second operand num2 using sc.nextInt().
        iv. Declare a variable result to store the calculated result and initialize it to 0.
        v. Use a switch statement based on the operator character:
            1. If the operator is '+', calculate the sum of num1 and num2 and assign it to result.
            2. If the operator is '-', calculate the difference of num1 and num2 and assign it to result.
            3. If the operator is '*', calculate the product of num1 and num2 and assign it to result.
            4. If the operator is '/', check if num2 is zero:
                a. If num2 is zero, throw a new ArithmeticException with the message "Exception: Division by Zero".
                b. Otherwise, calculate the quotient of num1 divided by num2 and assign it to result.
            5. If the operator is none of the above, throw a new Exception with the message "Exception: The operator is not valid".
        vi. Print the value of result.
    d. If any exception occurs within the try block, the control will be transferred to the catch block.
    e. In the catch block, catch the exception object e and print the error message using e.getMessage().
4. End the program.