**ALGORITHMS**

**1.PALINDROME CHECKER**

**Algorithm: CheckPalindrome**

**1. Start**

**2. Read a string 'str' from the user.**

**3. Convert 'str' to lowercase to make the palindrome check case-insensitive.**

**4. Initialize a variable 'flag' to 0.**

**5. Get the length of the string and store it in 'len'.**

**6. Iterate through the first half of the string using a loop (from i = 0 to len/2 - 1):**

**a. If the character at position 'i' is not equal to the character at position (len - i - 1),**

**set 'flag' to 1 and break out of the loop.**

**7. If 'flag' is 0, print "Palindrome"; otherwise, print "Not Palindrome".**

**8. End**

**2.FREQUENCY COUNTER**

**Algorithm: CountOccurrences**

**1. Start**

**2. Read a string 'str' from the user.**

**3. Convert 'str' to lowercase to make the character comparison case-insensitive.**

**4. Read a character 'ch' from the user.**

**5. Initialize a variable 'count' to 0.**

**6. Iterate through each character in the string 'str' using a loop:**

**a. If the current character is equal to 'ch', increment 'count' by 1.**

**7. Print the count of occurrences of 'ch' in 'str'.**

**8. End**

**3.**

**Algorithm: MatrixMultiplication**

**1. Start**

**2. Read the order of the first matrix (m1, n1) from the user.**

**3. Read the order of the second matrix (m2, n2) from the user.**

**4. If n1 is not equal to m2, print "Matrix Multiplication not Possible" and end.**

**5. Otherwise, continue to the next step.**

**6. Create matrices A, B, and C with dimensions m1 x n1, m2 x n2, and m1 x n2, respectively.**

**7. Read the elements of matrix A from the user.**

**8. Read the elements of matrix B from the user.**

**9. Perform matrix multiplication:**

**a. For each element C[i][j] in matrix C:**

**- Initialize C[i][j] to 0.**

**- Use a nested loop to iterate through elements of matrices A and B:**

**\* For each element k from 0 to n1-1:**

**\* C[i][j] += A[i][k] \* B[k][j]**

**10. Print matrices A, B, and C.**

**11. End**

**4.**

**Algorithm: EmployeeManagement**

**1. Define a class Employee with private attributes name, age, phone, address, and salary.**

**- Include methods printSalary, a parameterized constructor to initialize the attributes, and displayEmployee to print details.**

**2. Define a class Manager that extends Employee.**

**- Add private attributes specialization and department.**

**- Include a parameterized constructor to initialize all attributes, and displayManager to print Manager-specific details.**

**3. Define a class Officer that extends Employee.**

**- Add private attributes specialization and department.**

**- Include a parameterized constructor to initialize all attributes, and displayOfficer to print Officer-specific details.**

**4. Define a class emp for user input and testing.**

**a. Read Manager details from the user:**

**- Read name, age, phone, address, salary, specialization, and department.**

**- Create a Manager object using the provided details.**

**- Display Manager details using the displayManager method.**

**b. Read Officer details from the user:**

**- Read name, age, phone, address, salary, specialization, and department.**

**- Create an Officer object using the provided details.**

**- Display Officer details using the displayOfficer method.**

**5. End**

**5.**

**Algorithm: ShapeHierarchy**

**1. Define an abstract class named 'shape'.**

**a. Include an abstract method 'numberofsides()' that represents the number of sides of a shape.**

**2. Define a class 'rectangle' that extends 'shape'.**

**a. Implement the 'numberofsides()' method to print "Number of sides = 4".**

**3. Define a class 'triangle' that extends 'shape'.**

**a. Implement the 'numberofsides()' method to print "Number of sides = 3".**

**4. Define a class 'hexagon' that extends 'shape'.**

**a. Implement the 'numberofsides()' method to print "Number of sides = 6".**

**5. Define a class 'b' for testing the shape hierarchy.**

**a. Create objects of 'rectangle', 'triangle', and 'hexagon'.**

**b. Call the 'numberofsides()' method for each object to display the number of sides.**

**6. End**

**6.Algorithm: MemoryReleaseExample**

**1. Define a class named 'a'.**

**a. Include a finalize() method, which will be called by the garbage collector when the object is being reclaimed.**

**i. Print "Object memory is released" in the finalize() method.**

**2. In the main method of class 'a':**

**a. Create an object 't' of class 'a'.**

**b. Set 't' to null, indicating that there are no references to the object.**

**c. Explicitly request garbage collection by calling System.gc().**

**3. End**

**7.**

**8.**

**9.**

**10.**

**Algorithm: SynchronizedPrinting**

**1. Define a class named 'Display'.**

**a. Include a synchronized method 'print(String msg)' to print a message enclosed in square brackets.**

**b. Sleep for 1000 milliseconds (1 second) within the synchronized block.**

**2. Define a class named 'SyncThread' that extends 'Thread'.**

**a. Include private attributes 'd' of type 'Display' and 'msg' of type 'String'.**

**b. Include a parameterized constructor to initialize 'd' and 'msg'.**

**c. Override the 'run()' method to call the 'print()' method of the 'Display' object.**

**3. Define a class named 'test' for testing the synchronized printing.**

**a. Create an object 'd' of class 'Display'.**

**b. Create two 'SyncThread' objects 't1' and 't2' with different messages and the same 'Display' object.**

**c. Start both threads using the 'start()' method.**

**4. End**

**11.**

**Algorithm: SimpleCalculator**

**1. Define a class named 'husky' that extends 'JFrame' and implements 'ActionListener'.**

**a. Include private attributes for text field 't1', buttons 'b1' through 'b17', 'res', and 'operation'.**

**b. Initialize the UI components, buttons, and their positions in the constructor.**

**c. Implement the 'doAction' method to perform arithmetic operations based on the button pressed.**

**d. Implement the 'actionPerformed' method to handle button clicks and update the text field accordingly.**

**2. In the 'doAction' method:**

**a. Check if 'operation' is null.**

**- If true, set 'operation' to the current operation, store the current value in 'res', and clear the text field.**

**- If false, perform the arithmetic operation based on the current 'operation' and the value in the text field.**

**- If the operation is division ('/'), handle the case of dividing by zero and display an error message.**

**3. In the 'actionPerformed' method:**

**a. Check which button is clicked using 'e.getSource()'.**

**- If a numeric button, append the corresponding digit to the text field.**

**- If the clear button is clicked, clear the text field and reset 'res' and 'operation'.**

**- If an operation button is clicked, call the 'doAction' method with the corresponding operation.**

**- If the equals button is clicked, call the 'doAction' method with "=".**

**4. In the 'main' method:**

**a. Create an instance of the 'husky' class and make it visible.**

**5. End**

**12.**

**Algorithm: TrafficLightSimulation**

**1. Define a class named 'TrafficLight' that extends 'JPanel' and implements 'ActionListener'.**

**a. Include private attributes for radio buttons 'r1', 'r2', and 'r3', and colors 'red\_c', 'green\_c', 'orange\_c'.**

**b. Initialize UI components, radio buttons, colors, and add action listeners in the constructor.**

**c. Implement the 'actionPerformed' method to handle radio button selection and update colors accordingly.**

**d. Override the 'paintComponent' method to paint the traffic light circles based on selected colors.**

**2. In the 'actionPerformed' method:**

**a. Check which radio button is selected using 'r1.isSelected()', 'r2.isSelected()', and 'r3.isSelected()'.**

**b. Update the colors 'red\_c', 'green\_c', and 'orange\_c' based on the selected radio button.**

**c. Call 'repaint()' to refresh the UI.**

**3. In the 'paintComponent' method:**

**a. Draw three circles representing the traffic light using 'g.drawOval'.**

**b. Fill each circle with the corresponding color using 'g.setColor' and 'g.fillOval'.**

**4. Define a class named 'traffic' for testing the traffic light simulation.**

**a. Create a JFrame 'f1' with a size of 600x480 and set it to visible.**

**b. Create an instance of the 'TrafficLight' class, add it to the JFrame, and set the layout to null.**

**5. End**

**13.**

**Algorithm: linkedlist**

**1. Define a class named 'linkedlist'.**

**a. Include a private inner class named 'Node' with attributes 'data', 'left', and 'right'.**

**b. Include a method 'insert' to insert a new node with the given data at the end of the linked list.**

**c. Include a method 'delete' to delete the first node from the linked list.**

**d. Include a method 'display' to print the elements of the linked list.**

**2. In the 'insert' method:**

**a. Create a new node 'temp' with the given data.**

**b. If the linked list is empty (head is null), set 'head' to 'temp'.**

**c. Otherwise, traverse the linked list to the end and add 'temp' as the right child of the last node.**

**3. In the 'delete' method:**

**a. Retrieve the data of the first node ('x').**

**b. Update 'head' to be the right child of the current head.**

**c. Set the left child of the new head to null.**

**d. Print a message indicating that the element 'x' has been deleted.**

**4. In the 'display' method:**

**a. Check if the linked list is empty (head is null).**

**- If true, print "List is empty."**

**- If false, traverse the linked list and print the data of each node.**

**### Algorithm for the 'prgm' Class:**

**```plaintext**

**Algorithm: prgm**

**1. Define a class named 'prgm'.**

**a. Create a main method.**

**2. In the main method:**

**a. Create an instance of the 'linkedlist' class named 'list'.**

**b. Create a 'Scanner' object 'sc' for user input.**

**c. Initialize a string 'choice' to an empty string.**

**d. Use a while loop to repeatedly display a menu and process user choices until 'choice' is "4".**

**i. Display the menu with options for insert, delete, display, and exit.**

**ii. Prompt the user to enter a choice.**

**iii. Use a switch statement to perform actions based on the user's choice.**

**- If the choice is "1", prompt the user for data and call 'list.insert(data)'.**

**- If the choice is "2", call 'list.delete()'.**

**- If the choice is "3", call 'list.display()'.**

**- If the choice is "4", exit the loop.**

**- If the choice is invalid, print "Invalid Choice".**