Exercise No 1

Palindrome check

Aim: Write a Java program that checks whether a given string is a palindrome or not

Algorithm:

```
1. Start

2. Initialize a string variable `str`.

3. Initialize an integer variable `flag` and set it to 0.

4. Create a Scanner object `sc` for user input.

5. Print "Enter a string: ".

6. Read the input string from the user and store it in the variable `str`.

7. Calculate the length of the input string and store it in the variable `len`.

8. Convert the input string to lowercase using the `toLowerCase` method.

9. Initialize an integer variable `strLen` and set it to `len - 1`.

10. Start a for loop from `i = 0` to `i < len`.
    a. Check if the character at position `i` in the string is not equal to the
        character at position `strLen`.
       i. If true, set `flag` to 1 and break out of the loop.
       ii. If false, continue to the next iteration.
    b. Decrement `strLen` by 1.

11. Outside the loop, check if `flag` is equal to 1.
    a. If true, print "Not palindrome..!!".
    b. If false, print "Is a palindrome..!!".
```

Result:
Java program that checks whether a given string is a palindrome or not successfully obtained and output verified.

Exercise No 2

Frequency of character in string

Aim: Write a Java Program to find the frequency of a given character in a string.

Algorithm:

1. Initialize count to 0.

2. Create a Scanner object named "sc" to read input.

3. Print "Enter a string" to prompt the user.

4. Read a string from the user and store it in the variable "str".

5. Print "Enter a character: " to prompt the user.

6. Read a character from the user and store the first character in the variable "a".

7. Initialize a for loop with the loop variable "i" ranging from 0 to the length of the string "str".
   a. Check if the character at position "i" in the string "str" is equal to the character "a".
      i. If true, increment the "count" by 1.

8. Print the result indicating that the character "a" is repeated "count" times in the string.

9. End the program.

Result:

Java Program to find the frequency of a given character in a string successfully obtained and output verified.

Exercise No 3

Matrix multiplication

Aim: Write a Java program to multiply two given matrices. .

Algorithm:

```
1. Initialize variables:
   a. Declare variables: i, j, k, temp as integers.
   b. Declare variables: row2, column2 as integers.
   c. Declare arrays: mat1[][], mat2[][], result[][] as integer arrays of size 20x20.
   d. Create a Scanner object named 'sc' for user input.

2. Input the size of Matrix 1:
   a. Prompt the user to enter the size of Matrix 1 (row1/column1).
   b. Read and store the values of row1 and column1 from the user.

3. Input values for Matrix 1:
   a. Display a message asking for Matrix 1 values.
   b. Use nested loops to read and store the values of Matrix 1 (mat1[][]).

4. Input the size of Matrix 2:
   a. Use a do-while loop:
      i. Prompt the user to enter the size of Matrix 2 (row2/column2).
      ii. Read and store the values of row2 and column2 from the user.
      iii. Check if row2 is not equal to column1. If true, repeat steps i and ii.

5. Input values for Matrix 2:
   a. Display a message asking for Matrix 2 values.
   b. Use nested loops to read and store the values of Matrix 2 (mat2[][]).

6. Perform Matrix Multiplication:
   a. Use nested loops:
      i. Loop through rows of Matrix 1 (row1).
      ii. Loop through columns of Matrix 2 (column2).
      iii. Loop through the common dimension (column1).
         A. Calculate the dot product of corresponding row and column elements,
            storing the sum in 'temp'.
      iv. Assign 'temp' to the corresponding cell in the result matrix (result[][]).
      v. Reset 'temp' to 0 for the next iteration.
```

```
7. Display the Resultant Matrix:
   a. Print "Result :" to indicate the beginning of the result display.
   b. Use nested loops:
      i. Loop through rows of the result matrix (row1).
      ii. Loop through columns of the result matrix (column2).
         A. Print each element of the result matrix.
      iii. Print a newline after each row for formatting.

8. End of the program.
```

Result:

Java program to multiply two given matrices successfully obtained and output verified.

Exercise No 4

Inheritance

Aim: Write a Java program which creates a class named 'Employee' having the following members: Name, Age, Phone number, Address, Salary. It also has a method named 'print-Salary()' which prints the salary of the Employee. Two classes 'Officer' and 'Manager' inherits the 'Employee' class. The 'Officer' and 'Manager' classes have data members 'specialization' and 'department' respectively. Now, assign name, age, phone number, address and salary to an officer and a manager by making an object of both of these classes and print the same. (Exercise to understand inheritance)

Algorithm:

```
1. Start the program.
2. Define the `Employeee` class.
   a. Initialize Scanner object `sc`.
   b. Declare variables `name`, `address`, `phNumber`, `age`, and `salary`.
   c. Create a default constructor for `Employeee`.
      i. Display "Enter name:" and read input into `name`.
      ii. Display "Enter age:" and read input into `age`.
      iii. Consume the newline character.
      iv. Display "Enter Phone number:" and read input into `phNumber`.
      v. Display "Enter address:" and read input into `address`.
      vi. Display "Enter salary:" and read input into `salary`.
   d. Define a method `printSalary()` to print the salary.

3. Define the `Officer` class extending `Employeee`.
   a. Declare variables `specialization` and `department`.
   b. Create a default constructor for `Officer`.
      i. Consume the newline character.
      ii. Display "Enter specialization:" and read input into `specialization`.
      iii. Display "Enter department:" and read input into `department`.

4. Define the `Manager` class extending `Employeee`.
   a. Declare variables `specialization` and `department`.
   b. Create a default constructor for `Manager`.
      i. Consume the newline character.

  ii. Display "Enter specialization:" and read input into `specialization`.
```

```
        iii. Display "Enter department:" and read input into `department`.

5. Define the `EmployeeImplement` class.
    a. Define the `main` method.
        i. Create instances of `Officer` and `Manager` classes.
        ii. Display details of the officer and manager using `System.out.println`.

6. End the program.
```

Result:

Java Program to show inheritance successfully obtained and output verified.

Exercise No 5

Polymorphism

Aim: Write a java program to create an abstract class named Shape that contains an empty method named numberOfSides( ). Provide three classes named Rectangle, Triangle and Hexagon such that each one of the classes extends the class Shape. Each one of the classes contains only the method numberOfSides( ) that shows the number of sides in the given geometrical structures. (Exercise to understand polymorphism)

Algorithm:

```
1. Start

2. Define an abstract class named `Shape` with an abstract method
`numberOfSides()`.

3. Define a concrete class `Triangle` that extends `Shape`.
  a. Implement the `numberOfSides()` method to print "Triangle : 3".

4. Define a concrete class `Rectangle` that extends `Shape`.
  a. Implement the `numberOfSides()` method to print "Rectangle : 4".

5. Define a concrete class `Hexagon` that extends `Shape`.
  a. Implement the `numberOfSides()` method to print "Hexagon : 6".

6. Define a class named `Abstraction`.
  a. Define the `main` method.
    i. Create an instance of `Triangle` named `t`.
    ii. Call the `numberOfSides()` method on the `t` object.
    iii. Create an instance of `Rectangle` named `r`.
    iv. Call the `numberOfSides()` method on the `r` object.
    v. Create an instance of `Hexagon` named `h`.
    vi. Call the `numberOfSides()` method on the `h` object.
```

Result:

Java program to demonstrate Polymorphism successfully
obtained and output verified.

Exercise No 6

Garbage Collector

Aim: Write a Java program to demonstrate the use of garbage collector.

Algorithm:

1. Create a class named GarbageCollectorDemo:
   This class contains the main method, which serves as the entry point for the program.

2. Declare and instantiate two objects of MyClass:
   Two objects (obj1 and obj2) of the MyClass are created, and each object is initialised with a unique name.

3. Set the references to null:
   The references to the objects (obj1 and obj2) are set to null, indicating that there are no more references to these objects.

4. Invoke the garbage collector explicitly:
   The System.gc() method is called to suggest to the JVM to run the garbage collector.

5. Introduce a delay with Thread.sleep:
   The program pauses for 1000 milliseconds (1 second) to allow some time for the garbage collector to run.

6. Create the MyClass class:
   Define the MyClass class, which has a constructor to initialise the name attribute and print a message when an object is created. It also has a finalize method that prints a message when an object is being garbage collected.

Result:

Java program to demonstrate the use of garbage collector successfully obtained and output verified.

Exercise No 7

String Tokenizer

Aim: Write a Java program that reads a line of integers, and then displays each integer, and the sum of all the integers (Use String Tokenizer class of java.util).

Algorithm:

1. Display the message: "Enter a line of integers separated by spaces:"

2. Read a line of input from the user using System.console().readLine() and store it in the variable inputLine.

3. Create a new StringTokenizer object named tokenizer with the input line as its parameter.

4. Initialize an integer variable sum to 0.

5. Display the message: "Individual integers:"

6. Initialize a while loop with the condition tokenizer.hasMoreTokens().

7. Inside the loop:
   a. Get the next token from the tokenizer and store it in the variable token.
   b. Convert the token to an integer using Integer.parseInt() and store it in the variable number.
   c. Display the number.
   d. Add the number to the sum.

8. Exit the loop when there are no more tokens in the tokenizer.

9. Display the message: "Sum of all integers: " followed by the value of sum.

Result:

Java program to demonstrate the use of use String Tokenizer class successfully
obtained and output verified.

Exercise No 8

FILE HANDLING-READER/WRITER

Aim: Write a file handling program in Java with reader/writer.

Algorithm:

1. Open a FileWriter:
   Create a new FileWriter object named writer.
   Specify the file to write to, in this case, "input.txt".

2. Open a BufferedWriter:
   Create a new BufferedWriter object named bufferedWriter.
   Pass the writer object to the BufferedWriter constructor.

3. Write to the File:
   Use the write method of bufferedWriter to write the string "My name is Ashwin wilson" to the file.

4. Close the BufferedWriter and FileWriter:
   Call the close method on bufferedWriter and writer to release resources and ensure data is written.

5. Open a FileReader:
   Create a new FileReader object named reader.
   Specify the file to read from, in this case, "input.txt".

6. Open a BufferedReader:
   Create a new BufferedReader object named bufferedReader.
   Pass the reader object to the BufferedReader constructor.

7. Read from the File:
   Create a String variable named line to store each line read from the file.
   Use a while loop to read lines from the file until there are no more lines (null is encountered).
   Print each line using System.out.println(line).

8. Close the BufferedReader and FileReader:
   Call the close method on bufferedReader and reader to release resources and ensure proper closure.

Result:

File handling program in Java with reader/writer successfully obtained and output verified.

Exercise No 9

FILE COPY-FILE RELATED EXCEPTIONS

Aim: Write a Java program that read from a file and write to file by handling all file related exceptions.

Algorithm:

1. Initialize a string variable inputFileName with the value "input.txt".

2. Try the following block of code:
   a. Create a FileReader object named reader and pass inputFileName to its constructor.
   b. Create a BufferedReader object named bufferedReader and pass reader to its constructor.
   c. Create a FileWriter object named writer and pass the string "output.txt" to its constructor.
   d. Create a BufferedWriter object named bufferedWriter and pass writer to its constructor.
   e. Create a string variable line.
   f. Enter a while loop that continues as long as reading a line from bufferedReader does not return null.
   i. Inside the loop, write the current line to the bufferedWriter.
   g. Close the bufferedReader.
   h. Close the reader.
   i. Close the bufferedWriter.
   j. Close the writer.

3. If a FileNotFoundException occurs during the execution of the try block:
   a. Print an error message indicating that the file specified by inputFileName was not found.

4. If an IOException occurs during the execution of the try block:
   a. Print an error message indicating that an IOException occurred while reading or writing the file.
   b. Print the stack trace of the exception.

Result:
Java program that read from a file and write to file by handling all file related exceptions successfully obtained and output verified.

Exercise No 10

USAGE OF TRY, CATCH, THROWS AND FINALLY

Aim: Write a Java program that shows the usage of try, catch, throws and finally.

Algorithm:

1. Define a class named ThrowsClass.
2. Inside ThrowsClass, declare a method named throwException with a throws clause for ArithmeticException.
3. Inside throwException method, attempt to perform an arithmetic operation where result = 10/0, which will throw an ArithmeticException.
4. End ThrowsClass.
5. Define another class named Exception.
6. Inside Exception class, declare the main method that takes a String array as its parameter.
7. Inside the main method, enclose the code in a try block to handle potential exceptions.
8. Instantiate an object of ThrowsClass named t.
9. Invoke the throwException method on the ThrowsClass object t.
10. Catch any ArithmeticException that might be thrown during the execution of the try block.
11. Inside the catch block, print the exception information using System.out.println(e).
12. After the catch block, include a finally block.
13. Inside the finally block, print "All the Exceptions are handled" using System.out.println.

14. End the main method.

15. End the Exception class.

Result:

Java program that shows the usage of try, catch, throws and finally successfully obtained and output verified.

Exercise No 11

MULTI-THREADED PROGRAM

Aim: : Write a Java program that implements a multi-threaded program which has three threads. First thread generates a random integer every 1 second. If the value is even, second thread computes the square of the number and prints. If the value is odd the third thread will print the value of cube of the number

Algorithm:

1. Import the required Java library: java.util.Random.

2. Create a class X that implements the Runnable interface.
   a. Declare a static integer variable random in class X.
   b. Initialize a Random object rand.
   c. Implement the run method:
   i. Generate a random number between 0 (inclusive) and 25 (exclusive) using rand.nextInt(25).
   ii. Assign the generated random number to the static variable random.
   iii. Print the value of random.

3. Create a class Y that implements the Runnable interface.
   a. Implement the run method:
   i. Check if the static variable random in class X is even (X.random % 2 == 0).
   ii. If true, calculate the square of random using (int) Math.pow(X.random, 2) and print the result followed by a newline.

4. Create a class Z that implements the Runnable interface.
   a. Implement the run method:
   i. Check if the static variable random in class X is odd (X.random % 2 != 0).
   ii. If true, calculate the cube of random using (int) Math.pow(X.random, 3) and print the result followed by a newline.

5. Create a class Multithread with the main method.
   a. Initialize a loop from i=0 to i<10.
   i. Inside the loop:

- Create three Thread objects: objX for class X, objY for class Y, and objZ for class Z.

- Pause the execution of the current thread for 1000 milliseconds using Thread.sleep(1000).
- Start the three threads: objX.start(), objY.start(), and objZ.start().

Result:

Java program that implements a multi-threaded program successfully obtained and output verified.

Exercise No 12

THREAD SYNCHRONIZATION

Aim: Write a Java program that shows thread synchronization.

Algorithm:

1. Create a class named First that extends the Thread class.
   a. Define a method display within the class, taking a string parameter msg.
   i. Print "[" followed by the message.
   ii. Try to sleep for 1000 milliseconds.
   iii. Catch and handle any InterruptedException, printing the exception.
   iv. Print "]".

2. Create a class named Second that extends the Thread class.
   a. Declare a member variable ob of type First.
   b. Declare a member variable str1 of type String.
   c. Create a constructor that takes a First object (obj) and a String (str) as parameters.
   i. Assign obj to the ob variable.
   ii. Assign str to the str1 variable.
   iii. Start the thread.

3. Implement the run method in the Second class.
   a. Synchronize on the ob object.
   b. Call the display method of the First object (ob) with the str1 parameter.

4. Create a class named SynchronisationDemo.
   a. Define the main method that takes an array of strings as a parameter.
   i. Create an instance of the First class (F).
   ii. Create three instances of the Second class (s1, s2, and s3) with the First object (F) and different names ("Name1", "Name2", and "Name3") as parameters.

Result:

Java program that shows thread synchronization successfully obtained and output verified.

Exercise No 13

SIMPLE CALCULATOR-JAVA SWING

Aim: Write a Java program that works as a simple calculator. Arrange Buttons for digits and the + - * % operations properly. Add a text field to display the result. Handle any possible exceptions like divide by zero. Use Java Swing

Algorithm:

1. Initialize GUI Components:

   Create a class CalculatorSwingProgram that extends JFrame and

   implements ActionListener.

   Declare static variables f (JFrame) and l (JTextField).

   Initialize strings first, op, and second to empty.

   Create buttons for digits (0-9), operators (+, -, *, /), ".", "C" (cancel), and "=".

   Initialize an integer eq to 0 (equals not pressed).

2. Initialize GUI Design:

   Create a JFrame titled "calculator".

   Create a JTextField (l) with 16 columns, set it as non-editable.

   Create buttons and add them to a JPanel (p).

   Set the background color of the panel to blue.

   Add the panel to the frame (f), set frame size, default close operation, and make it visible.

3. Register Action Listeners:

Register the CalculatorSwingProgram class as the ActionListener for all buttons.

4. Handle Button Clicks:

   Implement the actionPerformed method to handle button clicks.

   Retrieve the action command using e.getActionCommand().

   If the command is a digit or ".", update the display text accordingly.

   If the command is "C", clear the input.

   If the command is "=", perform the calculation and display the result.

   If the command is an operator, handle it appropriately.

5. Update Display:

   Use the setText method to update the text in the JTextField (l) based on user input.

6. Perform Calculations:

   Implement logic to perform arithmetic calculations based on the operator selected.

   Catch exceptions for division by zero.

7. Handle Equals Press:

   If the equals button is pressed, set eq to 1 and update the values of first, op, and second accordingly.

8. Main Function:

   Create an instance of CalculatorSwingProgram to launch the calculator.

Result:

Java program that works as a simple calculator successfully obtained and output verified.

Exercise No 14

TRAFFIC LIGHT

Aim: : Write a Java program that simulates a traffic light. The program lets the user select one of three lights: red, yellow, or green. When a radio button is selected, the light is turned on, and only one light can be on at a time. No light is on when the program starts.

Algorithm:

1. Initialize TrafficLight class:
   a. Create a class named TrafficLight that extends JPanel and implements ActionListener.

2. Initialize instance variables:
   a. Declare private instance variables:
   JRadioButton r1, r2, r3
   Color red_c, yellow_c, green_c

3. Create constructor:
   a. Define a constructor for TrafficLight class:
   Set the panel bounds to (0, 0, 500, 480).
   Instantiate JRadioButtons r1, r2, and r3.
   Initialize red_c, yellow_c, and green_c to the background color.
   Create a ButtonGroup and add the radio buttons to it.
   Add radio buttons to the panel.
   Add action listener (this) to each radio button.

4. Handle radio button actions:
   a. Implement the actionPerformed method:
   Check which radio button is selected.
   Update red_c, yellow_c, and green_c colors accordingly.
   Call repaint() to update the panel.

5. Implement paintComponent:
   a. Override the paintComponent method:
   Call the superclass's paintComponent method.
   Draw three ovals representing traffic lights at positions (50, 50), (50, 110), and (50, 170).
   Set the color of each oval using red_c, yellow_c, and green_c.

Fill each oval with the corresponding color.

6. Initialize TrafficLightSimulator class:
   a. Create a class named TrafficLightSimulator.

7. Initialize main method:
   a. Define the main method in TrafficLightSimulator class.
   Create a JFrame object named f.
   Set the JFrame visibility to true.
   Set the size of the JFrame to (500, 480).
   Set the layout manager to null.
   Create an instance of the TrafficLight class named t.
   Add the TrafficLight panel to the JFrame.
   Set the default close operation for the JFrame.

Result:

Java program that simulates a traffic light successfully obtained and output verified.

Exercise No 15

JAVA DATABASE CONNECTIVITY

Aim: Write a Java program to display all records from a table using Java Database Connectivity (JDBC)

Algorithm:

1. Start
2. Load the driver
3. Establish connection
4. Create statement object
5. Execute SQl query and sore the result in ResultSet object
6. Process the result
7. Stop the connection
8. Stop

Result:

Java program to display all records from a table using Java Database Connectivity successfully obtained and output verified.

Exercise No 16

DOUBLY LINKED LIST

Aim: Write a Java program for the following: **
1) Create a doubly linked list of elements.
2) Delete a given element from the above list.
3) Display the contents of the list after deletion.

Algorithm:

1. Start

2. Define a class Node with fields data, next, and prev.

3. Define a class DLinkedList with a field head.

4. In the Node class, create a constructor to initialize data, prev, and next.

5. In the DLinkedList class, define a method addNode to add a node to the end of the list.

   a. Create a new Node with the given data.

   b. If the list is empty, set the new node as the head.

   c. If the list is not empty, traverse to the end and add the new node.

6. In the DLinkedList class, define a method removeNode to remove a node with a given data.

   a. If the list is empty, print "List empty" and return.

   b. If the head contains the data, adjust the pointers and update the head if needed.

c. If the data is not in the head, traverse the list to find the node with the given data.
d. If the data is not found, print "Data not found" and return.

e. Adjust the pointers to remove the node from the list.

7. In the DLinkedList class, define a method display to print the elements of the list.

   a. If the list is empty, print "Empty List."

   b. Traverse the list and print each node's data.

8. In the main method:

   a. Create an instance of DLinkedList called list.

   b. Enter an infinite loop for user interaction.

   c. Print the menu with options: "1. add Node", "2. remove Node", "3. exit".

   d. Take user input and switch based on the input.

   e. If the input is '1':

   i. Prompt the user for data.

   ii. Call addNode with the entered data.

   iii. Call display to show the updated list.

   f. If the input is '2':

   i. Prompt the user for data to be deleted.

   ii. Call removeNode with the entered data.

   iii. Call display to show the updated list.

g. If the input is '3', exit the program.

9. End.

Result:

Java program to implement doubly linked list successfully obtained and output verified.

Exercise No 17

QUICK SORT ALGORITHM

Aim: Write a Java program that implements Quick sort algorithm for sorting a list of names in

Algorithm:

1. Start

2. Define the Quicksort class with a static method quickSort that takes an array of strings names, an integer low, and an integer high.

3. If low is less than high, proceed with the following steps; otherwise, return.
   a. Initialize variables l to low, h to high, and p to low.
   b. Initialize a temporary string variable temp.
   c. Enter a while loop with the condition l < h.
   Inside the loop:
   Increment l while names[l] is less than or equal to names[p] and l is less than h.
   Decrement h while names[h] is greater than names[p].
   If l is still less than h, swap names[l] and names[h].
   d. Swap names[h], names[p], and temp.
   e. Recursively call quickSort with arguments names, 0, and h-1.
   f. Recursively call quickSort with arguments names, h+1, and high.

4. Define the main method.
   a. Create a Scanner object sc to read input from the console.
   b. Print "How many names do you have: ".
   c. Read an integer n from the user.
   d. Create an array of strings names with size n.
   e. Consume the newline character from the input.
   f. Use a loop to input names from the user and store them in the names array.
   Inside the loop:
   Print "Enter name [i+1]: ".
   Read a string from the user and store it in names[i].
   g. Call the quickSort method with arguments names, 0, and n-1.
   h. Print "Sorted list:".

    i. Use an enhanced for loop to print each string in the sorted names array.
    Inside the loop:
    Print the current string.

5. End

Result:

Java program that implements Quick sort algorithm successfully obtained and output verified.

Exercise No 18

BINARY SEARCH ALGORITHM

Aim: Write a Java program that implements the binary search algorithm.

Algorithm:

```
1.  Start

2.  Initialize variables: i, j, limit, beg, mid, end, item, temp

3.  Create an array 'arr' of size 'limit'

4.  Display "Enter size of the array:"

5.  Read and store the value of 'limit' from the user

6.  Display "Enter 'limit' numbers:"

7.  Read 'limit' integers from the user and store them in the 'arr' array

8.  Perform Bubble Sort on 'arr' in descending order:
    a. Set j = limit - 1
    b. Iterate while j >= 0
    i. Iterate i from 0 to j
    a. If arr[i] > arr[i + 1], swap arr[i] and arr[i + 1]
    c. Decrement j by 1

9.  Display "Sorted array:"

10.         Iterate i from 0 to limit
    a. Display arr[i]

11.         Display "Enter item to search"

12.         Read and store the value of 'item' from the user

13.         Set beg = 0, end = limit - 1, mid = (beg + end) / 2

14.         While (arr[mid] != item) and (beg <= end):
    a. If item < arr[mid], set end = mid - 1
    b. Else, set beg = mid + 1
    c. Update mid = (beg + end) / 2

15. If arr[mid] == item, then:
    a. Display "Element found at: " + (mid + 1)

16. Else, display "Element not found"

17. End
```

Result:

Java program that implements the binary search algorithm successfully obtained and output verified.