**Vikas Meneni Lab 3 – Assignment 1 NUID:002309537**

**1.Problem 1**

**Problem Description:**

**The task is to develop a program that validates credit card numbers using the Luhn algorithm, which checks the accuracy of credit card entries to prevent errors and fraud. The program prompts the user to enter a card number, verifies if it is valid by checking specific card prefixes (Visa starts with 4, MasterCard with 5, American Express with 37, Discover with 6), and applies the Luhn checksum process. This involves doubling every second digit from the right, adjusting sums for digits greater than 9, and adding all odd-position digits. If the total is divisible by 10, the card is valid. This problem highlights the importance of input validation and error-checking techniques, which are crucial in secure and reliable software development. By learning this method, one can apply similar validation logic to other problem domains that require data integrity and accuracy.**

**Analysis:**

**The code utilizes Luhn's algorithm as the main method for validating credit card numbers. This algorithm is structured to assess the validity of a number through a series of straightforward mathematical calculations. It processes the credit card number by doubling every second digit from the right side. If the doubled digit exceeds 9, the digits of the resulting number are added together. The program then computes the total sum of all processed digits—both those that were doubled and those that were not—and verifies if this total is divisible by 10. If it is, the card number is deemed valid; if not, it is considered invalid.**

**isValid(long number): This method checks if the credit card number is valid using the Luhn algorithm. It adds up the processed digits and checks if the total is divisible by 10.**

**sumOfDoubleEvenPlace(long number): This method calculates the sum of the digits in even positions (from the right) after doubling them. If doubling a digit gives a number greater than 9, it calls the getDigit(int number) method to sum its digits.**

**getDigit(int number): This method returns the sum of the digits of a number if it’s more than 9. If it’s a single digit, it simply returns that digit.**

**sumOfOddPlace(long number): This method sums the digits located at odd positions (from the right) in the credit card number.**

**prefixMatched(long number, int d): This method checks if the credit card number starts with a specific prefix (d). It uses the getPrefix(long number, int k) method to get the required prefix for comparison.**

**getSize(long d): This method counts the number of digits in a number by converting it to a string and checking its length.**

**getPrefix(long number, int k): This method retrieves the first k digits from the credit card number. If there are fewer digits than k, it returns the entire number.**

**One of the difficulties encountered while implementing the Luhn algorithm was understanding its specific rules regarding which digits to double and how to sum the results. Properly indexing the digits to ensure the correct ones are processed proved to be tricky. To address input validation challenges, such as filtering out non-numeric values or handling excessively large numbers, an effective alternative solution is to utilize regular expressions. This approach simplifies the validation process by quickly filtering input to ensure that the credit card number is in the correct format.**

**Source code:**

**package edu.northeastern.csye6200;**

**import java.util.Scanner;**

**public class LAB3P1 {**

**public static void main(String[] args) {**

**Scanner inputScanner = new Scanner(System.*in*);**

**System.*out*.print("Enter a credit card number as a long integer: ");**

**long cardNumber = inputScanner.nextLong();**

**if (*isValid*(cardNumber)) {**

**System.*out*.println("The card number " + cardNumber + " is valid.");**

**} else {**

**System.*out*.println("The card number " + cardNumber + " is invalid.");**

**}**

**inputScanner.close();**

**}**

**public static boolean isValid(long number) {**

**int totalSum = *sumOfDoubleEvenPlace*(number) + *sumOfOddPlace*(number);**

**return (totalSum % 10 == 0);**

**}**

**public static int sumOfDoubleEvenPlace(long number) {**

**int total = 0;**

**String numStr = Long.*toString*(number);**

**for (int i = numStr.length() - 2; i >= 0; i -= 2) {**

**int digit = Character.*getNumericValue*(numStr.charAt(i));**

**total += *getDigit*(digit \* 2);**

**}**

**return total;**

**}**

**public static int getDigit(int number) {**

**return (number < 10) ? number : (number / 10) + (number % 10);**

**}**

**public static int sumOfOddPlace(long number) {**

**int total = 0;**

**String numStr = Long.*toString*(number);**

**for (int i = numStr.length() - 1; i >= 0; i -= 2) {**

**int digit = Character.*getNumericValue*(numStr.charAt(i));**

**total += digit;**

**}**

**return total;**

**}**

**public static boolean prefixMatched(long number, int d) {**

**return *getPrefix*(number, *getSize*(d)) == d;**

**}**

**public static int getSize(long d) {**

**return Long.*toString*(d).length();**

**}**

**public static long getPrefix(long number, int k) {**

**String numStr = Long.*toString*(number);**

**return (numStr.length() < k) ? number : Long.*parseLong*(numStr.substring(0, k));**

**}**

**}**

**Screenshots of sample runs:**

**Screenshot 1 :-**

**A screenshot of a computer

Description automatically generated**

**Screenshot 2:-**

**A screenshot of a message

Description automatically generated**

**2.Problem 2**

**Problem Description:**

**The task involves creating a Java program that performs a series of operations on an integer array through four predefined methods: reversing the array, rotating its elements, sorting the array, and finding the largest element. Each method addresses a specific requirement: the reverseArray() method reverses the order of the elements and prints the result; the rotateArray() method shifts the array's elements to the right by a specified number of places, wrapping around elements that exceed the array's length; the sortArray() method organizes the array in ascending order; and the largestElement() method identifies and displays the largest value within the array. This program not only showcases basic array manipulation techniques but also highlights their practical applications in various fields, including data processing and algorithm design. Understanding these operations can enhance problem-solving skills and facilitate the handling of data structures in more complex programming tasks.**

**Analysis:**

**The code for the second problem uses basic array manipulation techniques to solve four tasks: reversing, rotating, sorting, and finding the largest element in an integer array. To reverse the array, it swaps elements from the start and end towards the middle. For rotating the array, a temporary array is used to shift elements by a specified number of places, ensuring correct positioning when elements wrap around. Sorting is achieved using Java’s built-in Arrays.sort() method, which sorts the array in ascending order. Finally, to find the largest element, a simple loop iterates through the array to compare each value and track the maximum.**

**Each task focuses on basic array operations without additional data structures, keeping the solution efficient and easy to understand. The code handles array boundaries during rotation by using modular arithmetic and allows for reuse of the input array across all methods.**

**reverseArray(int[]nums):**

**This method reverses the elements of the given array. It swaps the first element with the last, the second with the second last, and so on until all the elements are reversed**

**rotateArray(int[]nums,intplaces):  
This method rotates the array elements by the specified number of places to the right. The last elements move to the front, while the others shift to the right. The method uses a temporary array to store the rotated values and copies them back to the original array.**

**sortArray(int[]nums):  
This method sorts the array in ascending order using a built-in sorting algorithm. The sorted array is printed after completion.**

**largestElement(int[]nums):  
This method scans through the array to find the largest element. It starts with the first element and compares it with each subsequent element to determine the largest value.**

**One key challenge encountered in array manipulation was managing indices during reversal and rotation. Ensuring that elements are correctly positioned while respecting array bounds can be complex, particularly when elements need to wrap around during rotation. To overcome this difficulty, an effective alternative solution is to use Java's Collections API, which offers built-in methods for sorting and reversing arrays. This approach simplifies implementation and reduces code complexity, allowing developers to focus more on the logic rather than manual index management. By leveraging these built-in tools, the development process becomes more efficient and the code more readable.**

**Top of Form**

**Bottom of Form**

**Source Code:**

**package edu.northeastern.csye6200;**

**import java.util.Arrays;**

**public class LAB3P2 {**

**public static void main(String[] args) {**

**int[] givenarr = {1, 21, 13, 14, 9};**

**System.*out*.print("Input array: ");**

**for (int num : givenarr) {**

**System.*out*.print(num + " ");**

**}**

**System.*out*.println();**

***reverseArray*(givenarr);**

**System.*out*.print("Reversed Array: ");**

**for (int num : givenarr) {**

**System.*out*.print(num + " ");**

**}**

**System.*out*.println();**

***rotateArray*(givenarr, 2);**

**System.*out*.print("Rotated Array: ");**

**for (int num : givenarr) {**

**System.*out*.print(num + " ");**

**}**

**System.*out*.println();**

***sortArray*(givenarr);**

**System.*out*.print("Sorted Array: ");**

**for (int num : givenarr) {**

**System.*out*.print(num + " ");**

**}**

**System.*out*.println();**

***largestElement*(givenarr);**

**}**

**public static void reverseArray(int[] inputArray) {**

**for (int start = 0, end = inputArray.length - 1; start < end; start++, end--) {**

**int placeholder = inputArray[start];**

**inputArray[start] = inputArray[end];**

**inputArray[end] = placeholder;**

**}**

**}**

**public static void rotateArray(int[] inputArray, int rotationCount) {**

**int arrayLength = inputArray.length;**

**rotationCount = rotationCount % arrayLength;**

**if (rotationCount == 0) return;**

**int[] temp = new int[arrayLength];**

**for (int i = 0; i < arrayLength; i++) {**

**temp[(i + rotationCount) % arrayLength] = inputArray[i];**

**}**

**System.*arraycopy*(temp, 0, inputArray, 0, arrayLength);**

**}**

**public static void sortArray(int[] inputArray) {**

**Arrays.*sort*(inputArray);**

**}**

**public static void largestElement(int[] inputArray) {**

**int maxValue = inputArray[0];**

**for (int num : inputArray) {**

**if (num > maxValue) {**

**maxValue = num;**

**}**

**}**

**System.*out*.println("The largest element: " + maxValue);**

**}**

**}**

**Screenshots of sample runs:**

**A screenshot of a computer

Description automatically generated**

**3.Problem 3:**

**Source Code:**

**package edu.northeastern.csye6200;**

**import java.util.Scanner;**

**public class LAB3P3 {**

**public static void main(String[] args) {**

**Scanner userInput = new Scanner(System.*in*);**

**System.*out*.print("Enter the number of values: ");**

**int totalCount = userInput.nextInt();**

**int[] numberArray = new int[totalCount];**

**System.*out*.print("Enter the numbers: ");**

**for (int index = 0; index < totalCount; index++) {**

**numberArray[index] = userInput.nextInt();**

**}**

**if (*isConsecutiveFour*(numberArray)) {**

**System.*out*.println("The list has consecutive fours");**

**} else {**

**System.*out*.println("The list has no consecutive fours");**

**}**

**userInput.close();**

**}**

**public static boolean isConsecutiveFour(int[] values) {**

**for (int index = 0; index < values.length - 3; index++) {**

**if (values[index] == values[index + 1] &&**

**values[index] == values[index + 2] &&**

**values[index] == values[index + 3]) {**

**return true;**

**}**

**}**

**return false;**

**}**

**}**

**Screen shot 1 :-**

**A screenshot of a computer

Description automatically generated**

**Screen Shot 2:-**

****

**4.Problem 4:**

**Source Code:**

**package edu.northeastern.csye6200;**

**import java.util.Scanner;**

**public class LAB3P4 {**

**public static void main(String[] args) {**

**Scanner consoleInput = new Scanner(System.*in*);**

**System.*out*.print("Enter a series of strings (separated by commas): ");**

**String inputString = consoleInput.nextLine();**

**String[] items = inputString.split(",\\s\*");**

**String longestItem = *findLongestString*(items);**

**System.*out*.print("The longest string(s) are: ");**

**boolean first = true;**

**for (String item : items) {**

**if (item.length() == longestItem.length()) {**

**if (!first) {**

**System.*out*.print(", ");**

**}**

**System.*out*.print(item);**

**first = false;**

**}**

**}**

**System.*out*.println("\nThe longest string was: " + longestItem);**

**consoleInput.close();**

**}**

**public static String findLongestString(String[] a) {**

**String longestItem = "";**

**for (String currentItem : a) {**

**if (currentItem.length() > longestItem.length()) {**

**longestItem = currentItem;**

**}**

**}**

**return longestItem;**

**}**

**}**

**Screenshots of sample runs:**

**A screenshot of a computer

Description automatically generated**