CYSE 6200 Lab 2 – Assignment 1
Name: **Tejaswini Chavan**
NU ID: **002474826**

Problem 1 description:

The objective in this task is to verify a credit card number using predetermined guidelines.
To maintain the accuracy of their numbers, credit cards adhere to specific patterns.
A legitimate credit card number needs to:
have thirteen to sixteen digits.

Prefixes like 4 (Visa), 5 (MasterCard), 37 (American Express), or 6 (Discover) are good places t o start.

To solve a crucial component of this issue, my program applies the Luhn algorithm, a complex yet refined technique for validating credit card numbers.
As I worked on this assignment, I came to understand that creating a trustworthy first line ofdefe nse in financial transactions is more important than simply verifying figures.
What fascinates me about this topic is how a seemingly simple mathematical algorithm may prev ent costly errors.

Examine the following situations:

- When entering their card number, a consumer unintentionally swaps two digits (for exam ple, typing 1234 as 1334).
- When a card reader malfunctions, a digit is interpreted incorrectly.
  For fraudulent purposes, a random credit card number is created by someone.

Why This Issue Is Important:

As I worked on this solution, I learned that this validation mechanism is not limited to credit card use; it also supports other real-world applications.
The same ideas could be used for:

- Routing numbers for banks
- Government identification numbers
- Package tracking numbers.
- Digital payment systems

The difficulty lay not just in putting the algorithm into practice but also in building a solid syste m that can manage a range of edge circumstances and give consumers unambiguous feedback.

Analysis:

Algorithm Implementation and Overall Design: I took a systematic approach to implementing the credit card validation:

1. Core Validation Logic:

- First phase: Process even-positioned digits
    - Start from right side
    - Double each digit
    - Handle two-digit results by adding individual digits
- Second phase: Handle odd-positioned digits
- Final phase: Combine results and verify divisibility by 10

2. Design Principles:

- Used String manipulation for easier digit processing
- Implemented robust error checking
- Separated concerns into distinct methods
- Focused on maintainable, readable code

Difficulties Encountered

Algorithm Logic Challenges:
When constructing sumOfDoubleEvenPlace, I first had trouble with the even/odd position processing. We had to process from right to left, thus we had to be careful with the indexing.
had to make sure that doubled digits, which produced two-digit numbers, were handled correctly.

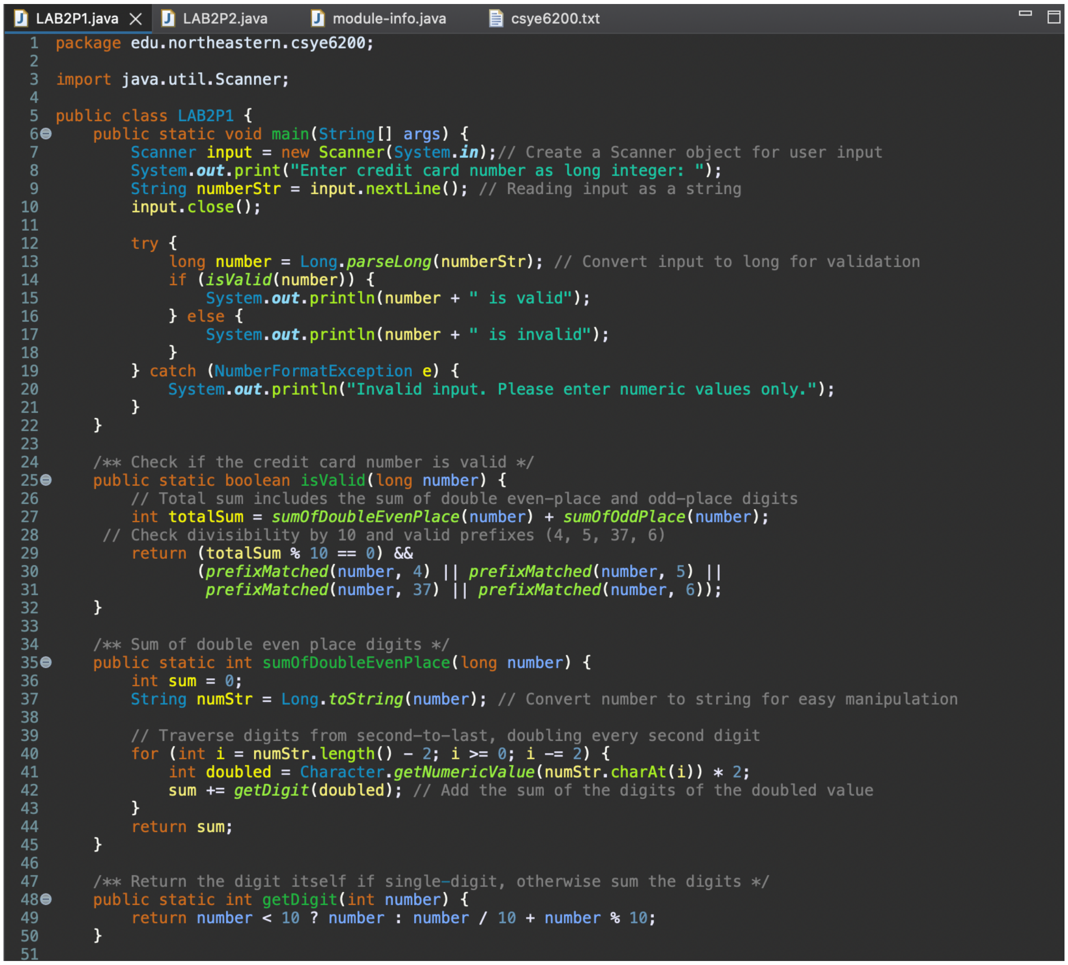
Problems with Technical Implementation:
Number integrity remains intact while converting between String and long data formats.
Making sure that edge cases (minimum/maximum card lengths) are handled correctly
Effectively implementing prefix matching without requiring numerous string conversions.

Future Improvements

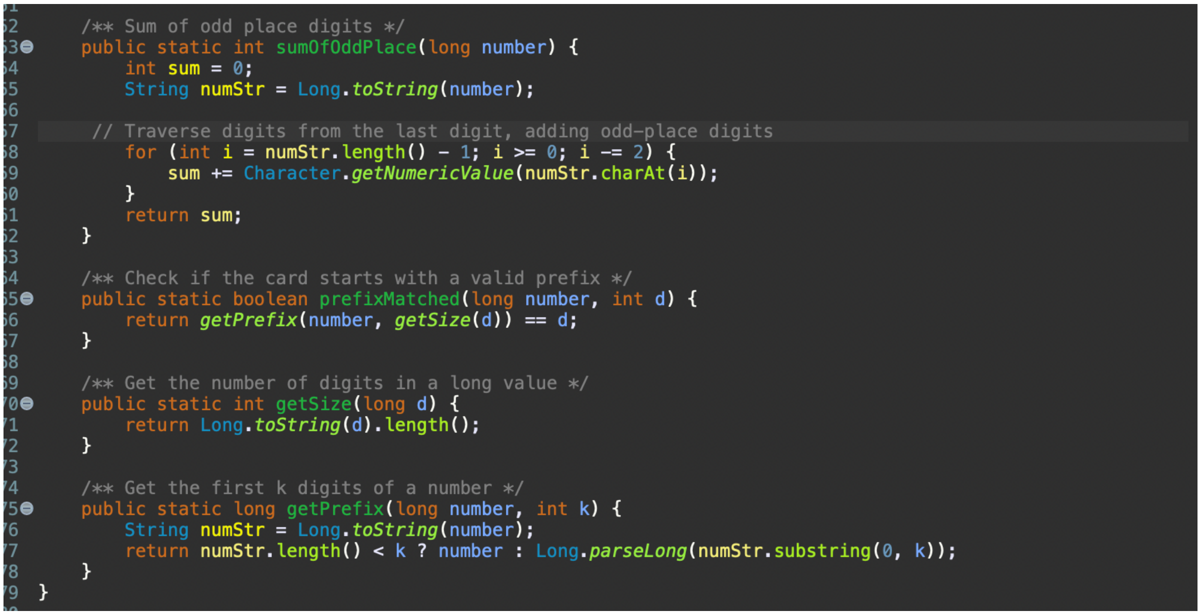
The following enhancements could be done with further time:

- Improved Error Handling: Incorporate a more thorough user feedback mechanism for incorrect inputs.
- Performance Optimization: While the software functions well for this use case, there is room for improvement when it comes to larger datasets.
- Enhancements to the User Experience: Create a more engaging interface that allows users to enter and verify several credit card numbers.

Source Code:

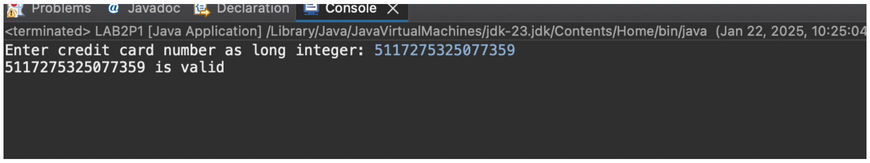


```java
package edu.northeastern.csye6200;

import java.util.Scanner;

public class LAB2P1 {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);// Create a Scanner object for user input
        System.out.print("Enter credit card number as long integer: ");
        String numberStr = input.nextLine(); // Reading input as a string
        input.close();

        try {
            long number = Long.parseLong(numberStr); // Convert input to long for validation
            if (isValid(number)) {
                System.out.println(number + " is valid");
            } else {
                System.out.println(number + " is invalid");
            }
        } catch (NumberFormatException e) {
            System.out.println("Invalid input. Please enter numeric values only.");
        }
    }

    /** Check if the credit card number is valid */
    public static boolean isValid(long number) {
        // Total sum includes the sum of double even-place and odd-place digits
        int totalSum = sumOfDoubleEvenPlace(number) + sumOfOddPlace(number);
     // Check divisibility by 10 and valid prefixes (4, 5, 37, 6)
        return (totalSum % 10 == 0) &&
               (prefixMatched(number, 4) || prefixMatched(number, 5) ||
                prefixMatched(number, 37) || prefixMatched(number, 6));
    }

    /** Sum of double even place digits */
    public static int sumOfDoubleEvenPlace(long number) {
        int sum = 0;
        String numStr = Long.toString(number); // Convert number to string for easy manipulation

        // Traverse digits from second-to-last, doubling every second digit
        for (int i = numStr.length() - 2; i >= 0; i -= 2) {
            int doubled = Character.getNumericValue(numStr.charAt(i)) * 2;
            sum += getDigit(doubled); // Add the sum of the digits of the doubled value
        }
        return sum;
    }

    /** Return the digit itself if single-digit, otherwise sum the digits */
    public static int getDigit(int number) {
        return number < 10 ? number : number / 10 + number % 10;
    }

```

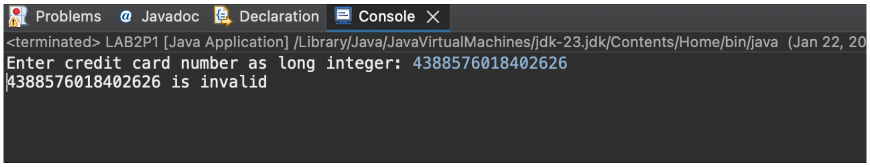


```java
    /** Sum of odd place digits */
    public static int sumOfOddPlace(long number) {
        int sum = 0;
        String numStr = Long.toString(number);

     // Traverse digits from the last digit, adding odd-place digits
        for (int i = numStr.length() - 1; i >= 0; i -= 2) {
            sum += Character.getNumericValue(numStr.charAt(i));
        }
        return sum;
    }

    /** Check if the card starts with a valid prefix */
    public static boolean prefixMatched(long number, int d) {
        return getPrefix(number, getSize(d)) == d;
    }

    /** Get the number of digits in a long value */
    public static int getSize(long d) {
        return Long.toString(d).length();
    }

    /** Get the first k digits of a number */
    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 run:



```
<terminated> LAB2P1 [Java Application] /Library/Java/JavaVirtualMachines/jdk-23.jdk/Contents/Home/bin/java  (Jan 22, 2025, 10:25:04
Enter credit card number as long integer: 5117275325077359
5117275325077359 is valid
```



```
<terminated> LAB2P1 [Java Application] /Library/Java/JavaVirtualMachines/jdk-23.jdk/Contents/Home/bin/java  (Jan 22, 20
Enter credit card number as long integer: 4388576018402626
4388576018402626 is invalid
```

Problem 2 description:

The objective of this task is to process student grades from a file and calculate how each grade deviates from the class average. This problem focuses on file handling in Java and performing statistical calculations on the data.
The program needs to:

- Read grades from a file named "csye6200.txt"
- Process up to the first 15 grades
- Calculate the average of these grades
- Display each grade's deviation from the average

What makes this problem interesting is its real-world applicability in educational settings. As I worked on this solution, I realized this type of analysis is crucial for:

- Understanding grade distributions in a class
- Identifying outliers in student performance
- Helping educators adjust their teaching methods
- Providing students with context for their performance

Analysis:

Algorithm Implementation and Overall Design:

- The grade analysis system was created with an organized plan that covered a number of importantt design and implementation facets.
- I created an ArrayList for dynamic grade storage and used the Scanner class for file proce ssing to guarantee effective file reading activities.Only the first 15 grades in the dataset w ere processed in accordance with the specifications.

Design Principles:

- With separate approaches managing the primary logic and average calculation independently, thedesign principles prioritized preserving a clear separation of concerns.
  I made sure that decimal output was formatted correctly and included strong error handling for file operations.
- To improve readability and maintainability, I kept the code structure tidy and the variable names descriptive throughout the development process.

Difficulties Encountered

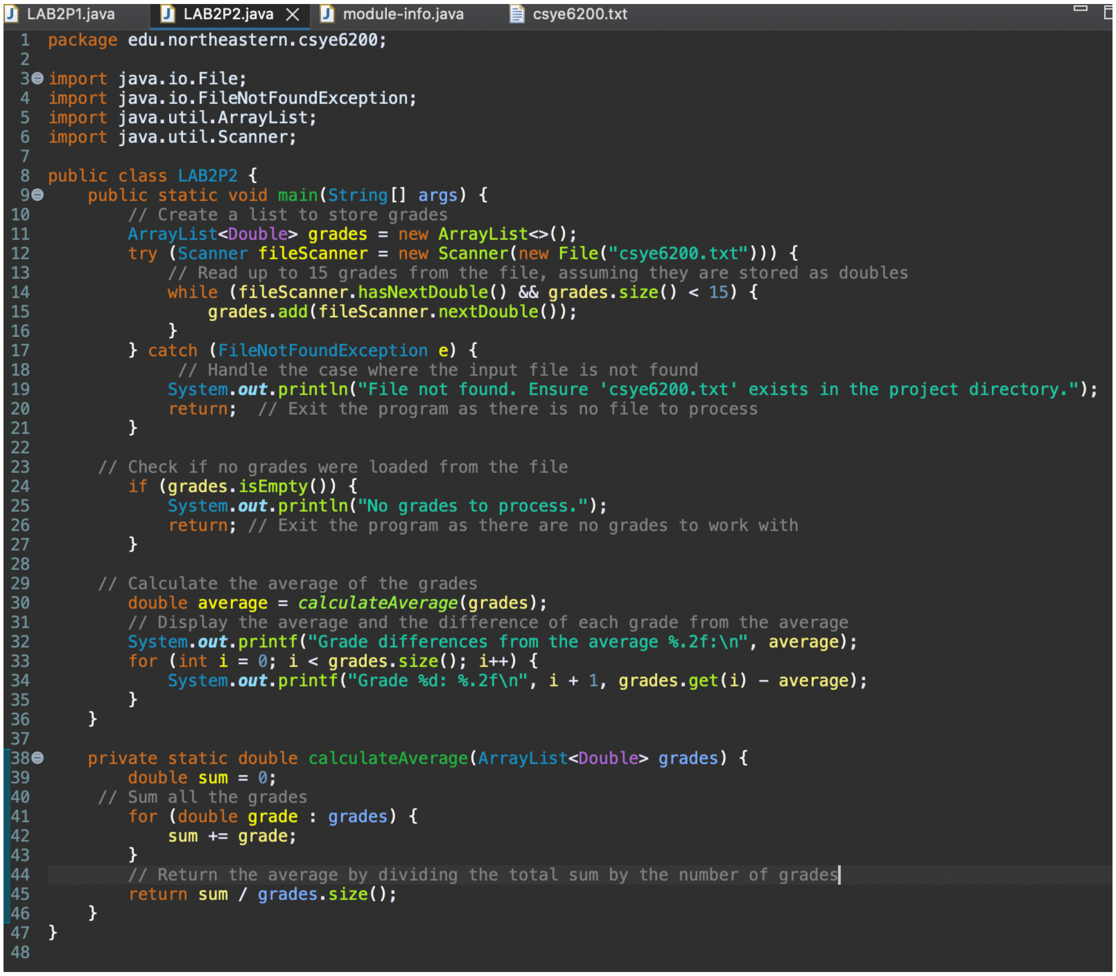
A number of difficulties surfaced in several areas during development. It was challenging to handle possible file format problems with non-numeric data, ensure correct file path resolution across various IDE settings, and efficiently manage file resource cleanup using try-with-resources blocks.

Data processing issues included maintaining precision in floating-point calculations, implementing suitable rounding for display purposes, and assuring accurate average computations with varied number of grades. To smoothly address missing file scenarios, empty file cases, and improper data types, error handling must be precise.

Future Improvements

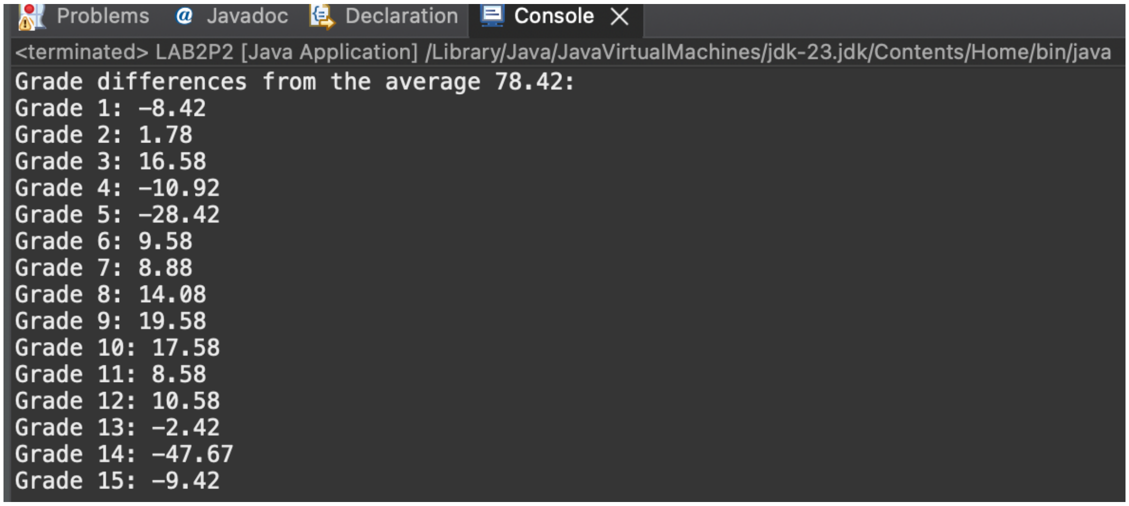
- Looking toward future improvements, several enhancements could be implemented given additional development time.
- Additional statistical measures including median, mode, and standard deviation, as well as visualrepresentations of grade distribution and grade categorization in relation to the class average, could be included in enhanced data analysis capabilities.
- File handling enhancements could enable batch processing for several class sections, enable dynamic file path specification, and extend support to multiple file formats, such as CSV and Excel.
- Enhancements to the user interface can include data export capabilities, options for various output formats, and interactive grade input possibilities.

Source code:



```java
package edu.northeastern.csye6200;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.Scanner;

public class LAB2P2 {
    public static void main(String[] args) {
        // Create a list to store grades
        ArrayList<Double> grades = new ArrayList<>();
        try (Scanner fileScanner = new Scanner(new File("csye6200.txt"))) {
            // Read up to 15 grades from the file, assuming they are stored as doubles
            while (fileScanner.hasNextDouble() && grades.size() < 15) {
                grades.add(fileScanner.nextDouble());
            }
        } catch (FileNotFoundException e) {
             // Handle the case where the input file is not found
            System.out.println("File not found. Ensure 'csye6200.txt' exists in the project directory.");
            return;  // Exit the program as there is no file to process
        }

     // Check if no grades were loaded from the file
        if (grades.isEmpty()) {
            System.out.println("No grades to process.");
            return; // Exit the program as there are no grades to work with
        }

     // Calculate the average of the grades
        double average = calculateAverage(grades);
        // Display the average and the difference of each grade from the average
        System.out.printf("Grade differences from the average %.2f:\n", average);
        for (int i = 0; i < grades.size(); i++) {
            System.out.printf("Grade %d: %.2f\n", i + 1, grades.get(i) - average);
        }
    }

    private static double calculateAverage(ArrayList<Double> grades) {
        double sum = 0;
     // Sum all the grades
        for (double grade : grades) {
            sum += grade;
        }
        // Return the average by dividing the total sum by the number of grades
        return sum / grades.size();
    }
}

```

Screenshots of Sample run:



```
<terminated> LAB2P2 [Java Application] /Library/Java/JavaVirtualMachines/jdk-23.jdk/Contents/Home/bin/java
Grade differences from the average 78.42:
Grade 1: -8.42
Grade 2: 1.78
Grade 3: 16.58
Grade 4: -10.92
Grade 5: -28.42
Grade 6: 9.58
Grade 7: 8.88
Grade 8: 14.08
Grade 9: 19.58
Grade 10: 17.58
Grade 11: 8.58
Grade 12: 10.58
Grade 13: -2.42
Grade 14: -47.67
Grade 15: -9.42
```