

SKILL DEVELOPMENT

Project Report



DLithe Consultancy Services Pvt. Ltd.

Project Report Assessment

Student Name: Vaishnavi Keshav Pattar

USN No: 2JR23CS116

Assignment: Java

Organization: DLithe Consultancy Services Pvt. Ltd.

Supervisor's Name: Archana SM

Submitted to

Signature of Training Supervisor

Signature of Students

Date:

Date:

Table of Contents

Introduction	4
Background	5
Sentence Capitalizer Program	6
Training Experience	12
Key Learnings	13
Challenges	15
Applications	16
Conclusion	17

Introduction:

The **Sentence Capitalizer Program** is a simple yet effective Java application that demonstrates fundamental string manipulation techniques. Its primary function is to take a sentence input from the user and return the same sentence with the first letter of each word capitalized and the remaining letters converted to lowercase. This type of text formatting is commonly used in applications such as title formatting, form input validation, and user interface enhancement.

The program makes use of standard Java libraries and follows a structured, modular approach. It begins by prompting the user for input using the Scanner class, which reads the entire sentence as a string. The string is then passed to a custom method named `capitalizeEachWord`, which is responsible for processing the sentence. This method trims the sentence, splits it into individual words, and applies capitalization to each word using basic string operations such as `substring()`, `toUpperCase()`, and `toLowerCase()`. The final result is then displayed back to the user in a neatly formatted output.

From a learning perspective, this program is particularly beneficial for beginners in Java programming. It introduces important concepts such as method creation, string manipulation, input/output handling, and control flow using loops and conditionals. Furthermore, it reinforces best practices such as input validation, code reusability, and modular design.

In real-world scenarios, capitalizing words in a sentence is a frequent requirement, especially in applications that deal with human-readable text. By developing this program, students and novice programmers gain valuable insight into how to process and format user input effectively.

Overall, the Sentence Capitalizer Program is a practical example of how simple logic can be used to solve common problems in software development, while simultaneously enhancing a programmer's understanding of core Java concepts.

Background:

In many software applications, especially those involving text processing, data formatting plays a crucial role in enhancing readability and ensuring consistency. One common formatting task is capitalizing the first letter of each word in a sentence, often referred to as “title case” or “capital case.” This is widely used in documents, user interfaces, data entry forms, headlines, and content management systems. The Sentence Capitalizer Program was developed as a response to this fundamental need for automated text formatting.

The idea behind the program is rooted in basic principles of programming and string manipulation. In languages like Java, manipulating strings requires a good understanding of how to extract, modify, and reassemble substrings. The Sentence Capitalizer Program provides a practical exercise for learning how to use string functions such as `split()`, `substring()`, `toUpperCase()`, and `toLowerCase()` in combination with loops and conditionals.

Additionally, the program demonstrates good coding practices such as modular design by encapsulating the capitalization logic in a separate method (`capitalizeEachWord`). It also uses input handling through the `Scanner` class, showcasing how to interact with users in console-based Java applications.

This program is often introduced in beginner to intermediate Java programming courses as it covers multiple core concepts in a compact and meaningful exercise. It encourages learners to think about string handling, edge cases (like multiple spaces or punctuation), and the importance of readable and maintainable code.

Overall, the Sentence Capitalizer Program is not just a utility tool but also an educational example that bridges theoretical understanding with practical coding skills. It reinforces fundamental programming techniques while providing a useful real-world application of text processing in Java.

Sentence Capitalizer Program Requirement Collection:

Project Overview: The "Sentence Capitalizer" is a Java console application designed to read a

sentence input from the user and return the same sentence with the first letter of each word capitalized. This program demonstrates string manipulation, use of methods, and basic Java input/output.

Functional Requirements:

- **Input:** The program should prompt the user to enter a sentence.
- **Processing:** The program will process the sentence by:
 - Splitting the sentence into words based on whitespace.
 - Capitalizing the first letter of each word.
 - Lowercasing the rest of each word.
 - Rejoining the words into a single sentence.
- **Output:** The program should display the capitalized sentence.

Non-Functional Requirements:

- The program must be implemented in Java.
- It should run in a command-line or terminal environment.
- The code must be clean and readable, with meaningful method names.
- Proper use of Java's built-in classes such as Scanner and StringBuilder.

System Requirements:

- Java Development Kit (JDK) version 8 or higher.
- Any text editor or IDE (e.g., IntelliJ IDEA, Eclipse, VS Code).
- Command-line interface (Terminal, Command Prompt) for execution.

User Requirements:

- The user must be able to enter any sentence containing alphabets and spaces.
- The user should see the capitalized version of their input clearly displayed.

Assumptions and Constraints:

- Input will be a single line of text.
- All input words are assumed to be separated by one or more spaces.

Input/Output Examples:

Input

"hello world"

" multiple spaces here "

"JAVA is FUN"

Output

"Hello World"

"Multiple Spaces Here"

"Java Is Fun"

Program Features Summary:

Feature

Sentence Input

Capitalization Logic

Output Display

Input Validation

Clean Exit

Description

Prompts and reads user input via Scanner

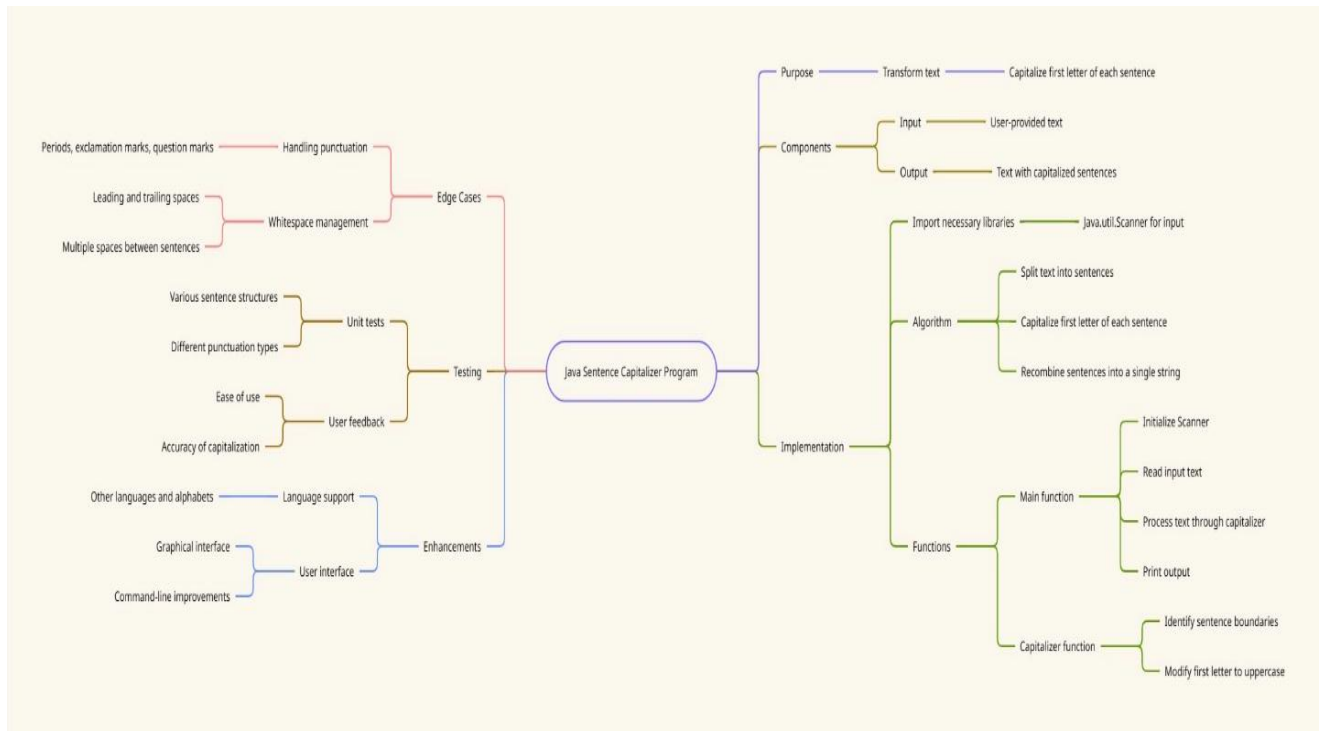
Custom method to capitalize each word

Displays the modified sentence

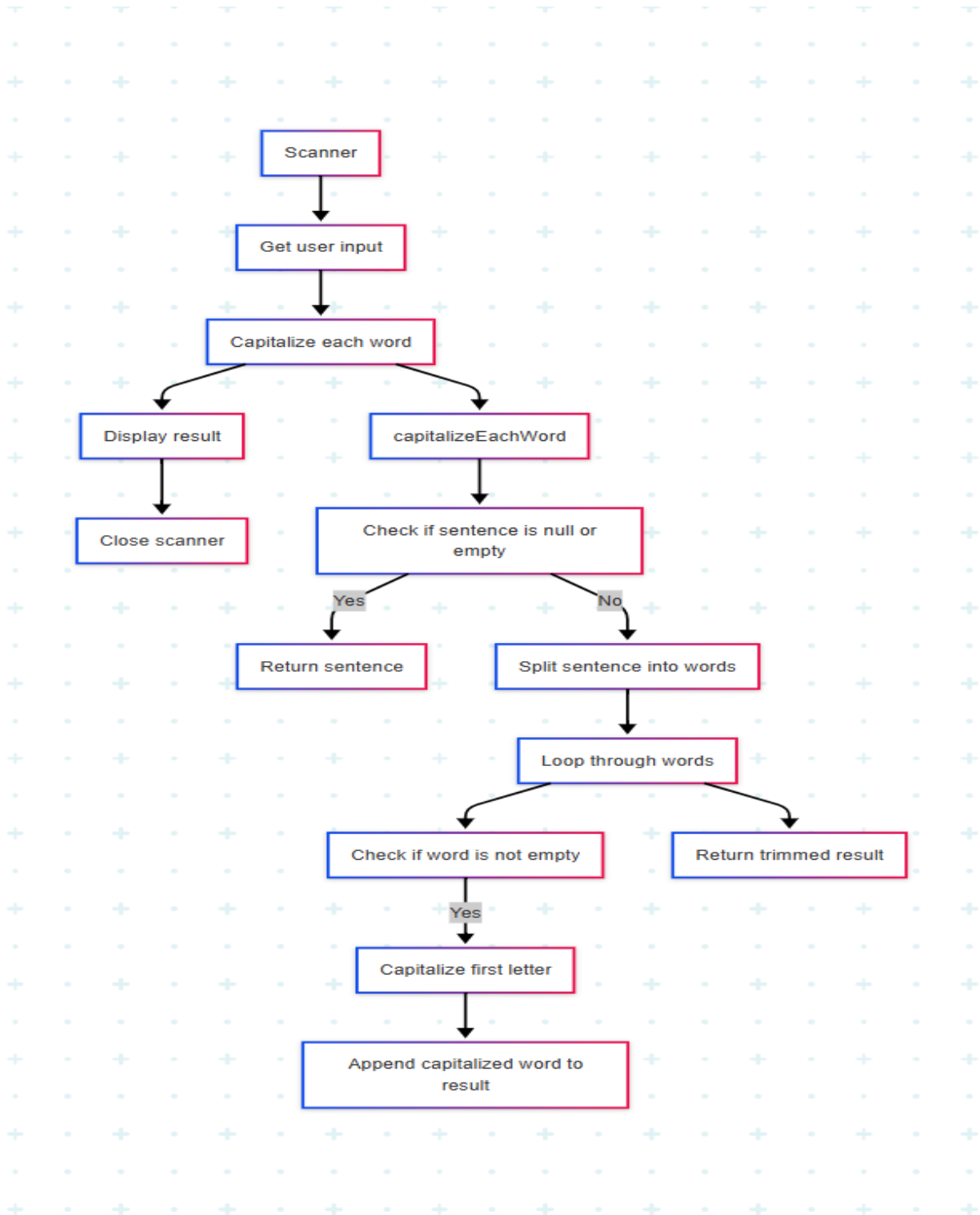
Checks for null or empty string

Closes scanner object

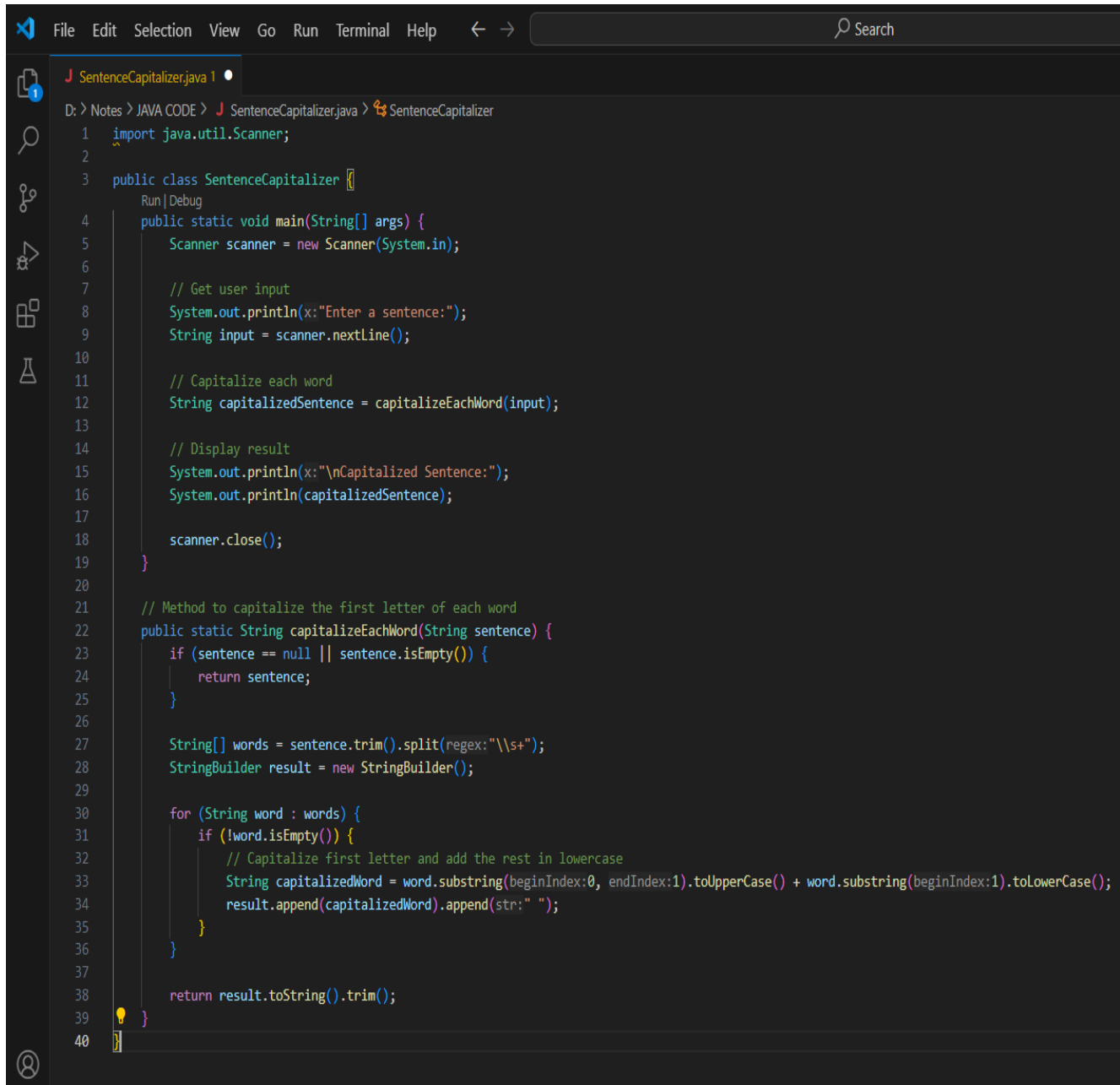
Mind Map:



Flowchart:



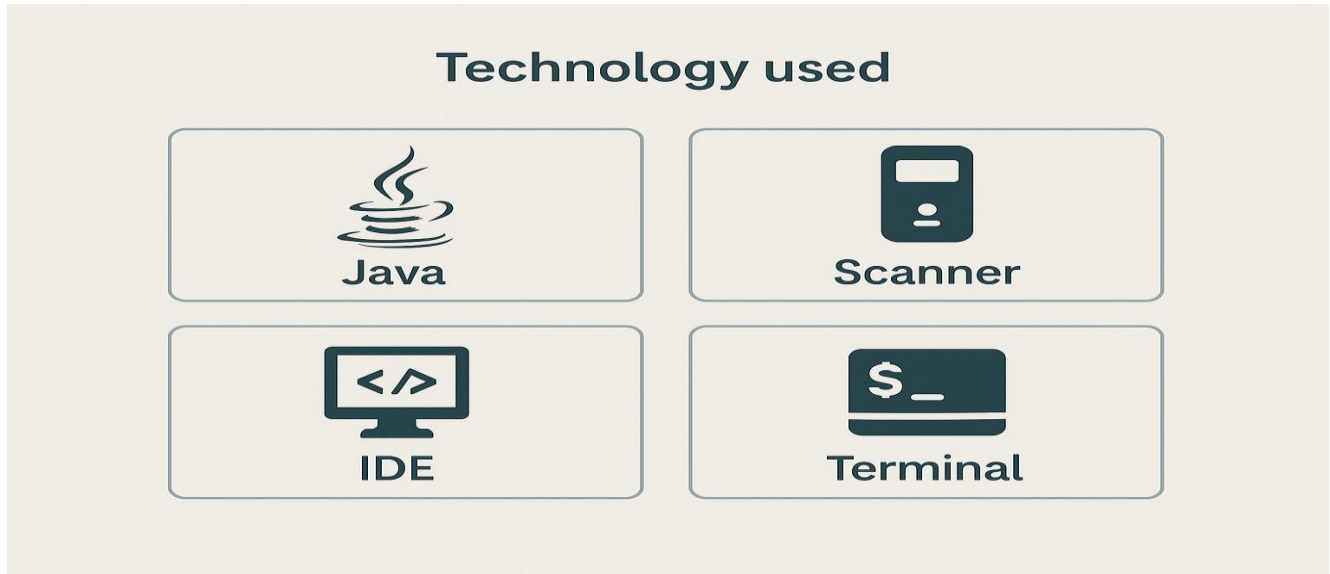
Project Development Images:



The screenshot shows an IDE with a dark theme. The top menu bar includes File, Edit, Selection, View, Go, Run, Terminal, and Help. A search bar is on the right. The left sidebar contains icons for Explorer, Search, Source Control, Run and Debug, Extensions, and Testing. The main editor area displays the code for `SentenceCapitalizer.java`. The code imports `java.util.Scanner` and defines a `SentenceCapitalizer` class with a `main` method and a `capitalizeEachWord` method. The `main` method prompts the user for a sentence, reads it, and prints the capitalized version. The `capitalizeEachWord` method uses `String.split` and `StringBuilder` to process the sentence.

```
1 import java.util.Scanner;
2
3 public class SentenceCapitalizer {
4     public static void main(String[] args) {
5         Scanner scanner = new Scanner(System.in);
6
7         // Get user input
8         System.out.println("Enter a sentence:");
9         String input = scanner.nextLine();
10
11        // Capitalize each word
12        String capitalizedSentence = capitalizeEachWord(input);
13
14        // Display result
15        System.out.println("\nCapitalized Sentence:");
16        System.out.println(capitalizedSentence);
17
18        scanner.close();
19    }
20
21    // Method to capitalize the first letter of each word
22    public static String capitalizeEachWord(String sentence) {
23        if (sentence == null || sentence.isEmpty()) {
24            return sentence;
25        }
26
27        String[] words = sentence.trim().split(regex:"\\s+");
28        StringBuilder result = new StringBuilder();
29
30        for (String word : words) {
31            if (!word.isEmpty()) {
32                // Capitalize first letter and add the rest in lowercase
33                String capitalizedWord = word.substring(beginIndex:0, endIndex:1).toUpperCase() + word.substring(beginIndex:1).toLowerCase();
34                result.append(capitalizedWord).append(str: " ");
35            }
36        }
37
38        return result.toString().trim();
39    }
40 }
```

Technologies used:



Output:

```
PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS
Debug: SentenceCapitalizer + v [ ] [ ] ... ^ X

PS C:\Users\Vaishnavi> & 'C:\Program Files\Java\jdk-24\bin\java.exe' '-agentlib:jdwp=transport=dt_socket,server
=n,suspend=y,address=localhost:55278' '--enable-preview' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Use
rs\Vaishnavi\AppData\Local\Temp\vscodesws_7f462\jdt_ws\jdt.ls-java-project\bin' 'SentenceCapitalizer'
Enter a sentence:
hello, this is my java mini project!!

Capitalized Sentence:
Hello, This Is My Java Mini Project!!
PS C:\Users\Vaishnavi> |
```

Training Experience:

Learning Java has been an enriching experience that introduced me to the core concepts of programming and object-oriented design. Through hands-on practice, I learned how to write, debug, and structure Java programs using classes, methods, and control structures.

The training helped me understand key concepts like inheritance, encapsulation, and string manipulation. I worked on small projects like the **Sentence Capitalizer**, which improved my coding and problem-solving skills.

Using tools like Eclipse and the Java API, I became more confident in handling input/output operations and building interactive programs. Overall, the experience has strengthened my programming foundation and prepared me for real-world application development.

Key Learnings:

String Manipulation Techniques:

Gained hands-on experience with Java string methods like `split()`, `substring()`, `toUpperCase()`, and `toLowerCase()` for transforming text.

User Input Handling:

Learned to take real-time user input using the `Scanner` class and handle empty or invalid input gracefully.

Modular Programming:

Practiced writing clean, reusable code by organizing logic into separate methods such as `capitalizeEachWord()`.

Control Structures:

Applied `if` conditions and `for` loops to iterate through arrays and apply transformations effectively.

Error Handling and Edge Case Management:

Understood the importance of checking for null or empty strings and trimming input to avoid runtime issues.

Basic Object-Oriented Concepts:

Reinforced knowledge of methods, objects, and encapsulation in Java through a simple but functional project.

Debugging and Testing Skills:

Improved ability to identify logic errors, test different input scenarios, and validate output correctness.

Console-Based Java Application Development:

Developed confidence in building interactive command-line programs with user-friendly input and output prompts.

Documentation and Code Comments:

Recognized the importance of writing clear comments and maintaining readable code for future reference and collaboration.

Project-Based Learning:

Realized the value of applying theoretical knowledge through projects to reinforce understanding and build confidence.

Challenges:

Handling Edge Cases in User Input:

Dealing with inputs that were empty, contained multiple spaces, or included special characters required additional checks and robust string handling logic.

Understanding String Manipulation Methods:

Initially, it was challenging to choose the right combination of methods (`substring()`, `toUpperCase()`, `toLowerCase()`, etc.) to correctly format each word.

Splitting Sentences into Words:

Learning to use regular expressions like `split("\\s+")` for separating words in a way that ignores multiple spaces took some practice.

Maintaining Consistent Formatting:

Ensuring that each word was capitalized correctly while the rest of the letters were in lowercase required careful attention to string slicing and character casing.

Avoiding Logic Errors:

Minor mistakes, such as indexing errors or forgetting to trim whitespace, led to incorrect output or runtime errors.

Reusability and Method Design:

Designing the `capitalizeEachWord()` method to be reusable and independent of the main program flow required modular thinking and proper parameter use.

Testing with Diverse Inputs:

Testing the program with different types of sentences (including numbers, punctuation, and non-alphabetic characters) exposed unexpected bugs and helped improve code robustness.

Building a User-Friendly Interface:

While the program is console-based, ensuring clear prompts and output formatting was important for usability.

Applications:

Text Editors and Word Processors:

Useful for formatting text automatically in tools like word processors, note-taking apps, or simple editors where consistent capitalization is needed.

Web Form Validation:

Enhances user input in online forms by automatically capitalizing names, addresses, or titles before storing them in a database.

Mobile Applications:

Can be integrated into Android apps (Java-based) for formatting user-generated content like comments, messages, or profile info.

Educational Tools:

Acts as a simple learning module for beginners to understand string manipulation, input/output operations, and clean code practices.

Data Cleaning for Databases:

Helps in preprocessing text data before inserting it into databases, ensuring consistency in capitalized fields like names, cities, etc.

Voice-to-Text Processing:

In speech recognition systems, it can help post-process the transcribed text to ensure proper capitalization for readability.

Report Generation:

Useful in automated systems that generate reports or documents where headings or section titles need consistent capitalization.

Command-Line Utilities:

Can be extended into a command-line tool for batch processing of multiple sentences or text files.

Conclusion:

The **Sentence Capitalizer Program** serves as a practical and educational introduction to Java programming, focusing on string manipulation, user input handling, and modular code design. Through the development of this project, I gained a solid understanding of how to process and transform user input, implement logical structures, and apply object-oriented principles in a real-world scenario. This project not only enhanced my technical skills in Java but also improved my ability to write clean, efficient, and reusable code. It reinforced the importance of problem-solving, debugging, and code readability—skills that are essential for any software developer. Overall, the Sentence Capitalizer Program has been a valuable learning experience, laying the foundation for more complex programming tasks and motivating further exploration into Java and software development.

