Green University of Bangladesh

Department of Computer Science and Engineering(CSE)

Faculty of Sciences and Engineering

Semester: (Spring, Year: 2025), B.Sc. in CSE (Day)

Lab Report NO 01

Course Title: Data Communication Lab

Course Code: CSE 308 Section: 223 D1

Student Details

Name	ID
Anisur Rahaman Maruf	222902078

Submission Date: 02/03/2025

Course Teacher's Name: Md. Samin Hossain Utsho

[For Teachers use only: Don't Write Anything inside this box]

Lab Report Status

Marks: …………………………………	Signature:.....................
Comments:..	Date:.............................

1. Title of the Experiment

Implementation of Byte Stuffing Algorithm in Java

2. Objectives/Aim

- To implement a byte stuffing algorithm in Java.
- To understand how special flag and escape sequences are handled in data transmission.
- To ensure data integrity by preventing misinterpretation of flag sequences.

3. Procedure / Analysis / Design

Step 1: Understanding Byte Stuffing

Byte stuffing is a process where an escape sequence is inserted before a predefined flag sequence or escape sequence found in the input data. In this experiment:

- The FLAG sequence is **"GALF"**.
- The ESCAPE sequence is **"EPACSE"**.

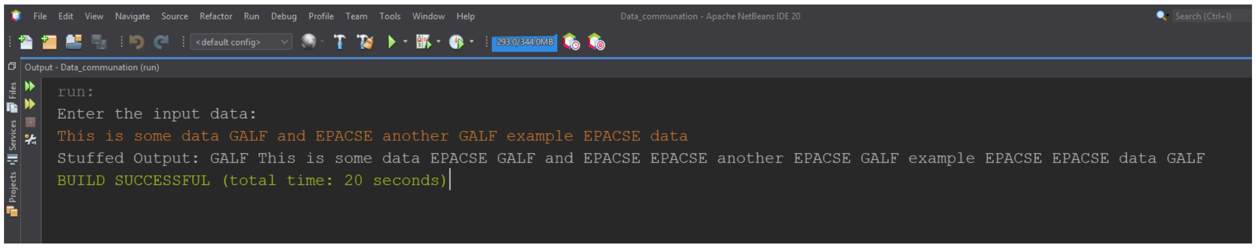
Step 2: Implementation Approach

- Read the input string from the user.
- Traverse the input string and check for occurrences of FLAG ("GALF") and ESCAPE ("EPACSE").
- If either sequence is found, prepend it with the ESCAPE sequence ("EPACSE").
- Construct and print the modified stuffed string.

Step 3: Java Implementation

```
import java.util.Scanner;

public class ByteStuffing {
    private static final String FLAG = "GALF";
    private static final String ESCAPE = "EPACSE";

    public static String performByteStuffing(String input) {
        StringBuilder stuffedData = new StringBuilder();
        stuffedData.append(FLAG);

        String[] words = input.split(" ");
        for (String word : words) {
            if (word.equals(FLAG) || word.equals(ESCAPE)) {
                stuffedData.append(" ").append(ESCAPE);
            }
            stuffedData.append(" ").append(word);
        }

        stuffedData.append(" ").append(FLAG);
        return stuffedData.toString().trim();
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter the input data:");
        String inputData = scanner.nextLine();

        String stuffedData = performByteStuffing(inputData);
        System.out.println("Stuffed Output: " + stuffedData);

        scanner.close();
    }
}
```

4. Output


File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Data_communation - Apache NetBeans IDE 20
<default config>
Output - Data_communation (run)

```
run:
Enter the input data:
This is some data GALF and EPACSE another GALF example EPACSE data
Stuffed Output: GALF This is some data EPACSE GALF and EPACSE EPACSE another EPACSE GALF example EPACSE EPACSE data GALF
BUILD SUCCESSFUL (total time: 20 seconds)
```

5. Discussion

- The algorithm effectively identifies FLAG and ESCAPE sequences and ensures they are properly prefixed with the ESCAPE sequence.
- This prevents accidental misinterpretation of data as a control sequence during transmission.
- The split(" ") method is used to process the data word by word, ensuring correct insertion of escape sequences.
- The final output is framed with FLAG sequences at the start and end to indicate message boundaries.

6. Conclusion

- The experiment successfully demonstrates the byte stuffing algorithm in Java.
- The implemented approach ensures correct handling of FLAG and ESCAPE sequences.
- Byte stuffing is a crucial concept in data communication, ensuring that control sequences are not confused with actual data.
- This implementation follows the exact byte stuffing mechanism shown in the given example.