EAST WEST UNIVERSITY

Department of Computer Science and Engineering

CSE 350

PROJECT

Submitted To

Md. Mahir Ashhab

Lecturer
Department of Computer Science & Engineering

Submitted By

Arif Mahmud Rakib

2019-1-60-070

Implement B8ZS and HDB3

Theoretical Background:

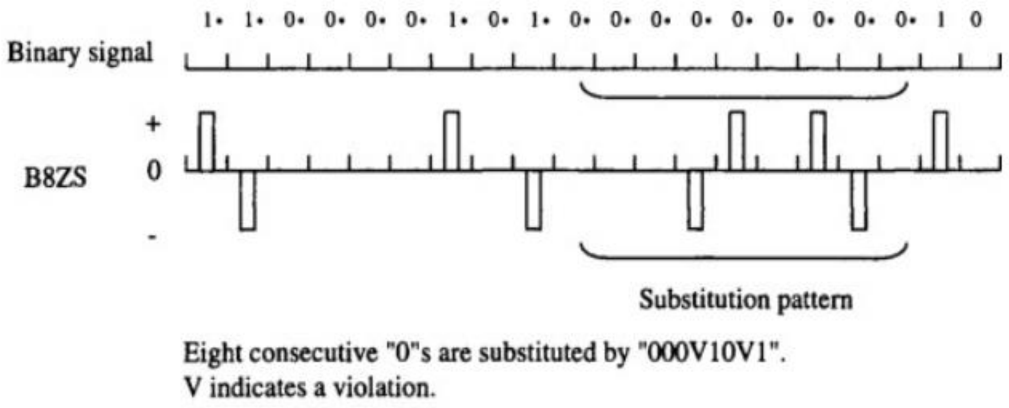
1. B8ZS (Bipolar with 8 Zeros Substitution):

- B8ZS is a line coding scheme used in telecommunications to ensure a balance of positive and negative pulses in the transmitted signal.

- When there are eight consecutive zeros in the data stream, B8ZS substitutes the pattern with a specific sequence of symbols to maintain signal integrity.

- The substituted pattern can be either \`+0-0-0-0+\` or \`-0+0+0+0-\`, depending on the previous state of the signal.

- The primary goal of B8ZS is to provide a sufficient number of transitions for clock recovery and to minimize the occurrence of long runs of zeros.



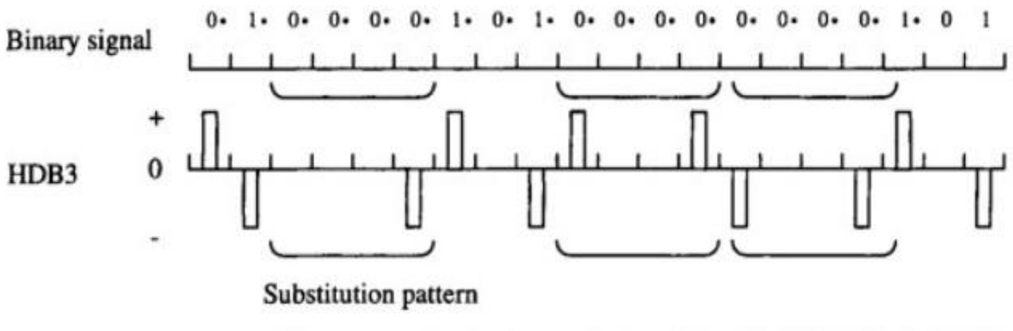
(a) B8ZS coding

2. HDB3 (High-Density Bipolar 3 Zeros):

- HDB3 is another line coding method used in telecommunications.

- Similar to B8ZS, HDB3 aims to maintain a balance of positive and negative pulses and limit the number of consecutive zeros in the signal.

- HDB3 replaces sequences of four consecutive zeros with specific patterns, which can be either `000V` or `B00V`, where `V` represents a violation and `B` denotes a bipolar violation.



Four consecutive "0"s are substituted by either "000V" or "100V".

(b) HDB3 coding

Implementation Details:

1. Encoding Logic:

- For B8ZS, you'll need to implement logic to detect sequences of eight consecutive zeros and substitute them with the appropriate pattern.

- For HDB3, you'll need to detect sequences of four consecutive zeros and replace them with the appropriate pattern.

- Both methods require maintaining the state of the signal to determine the correct substitution.

2. Decoding Logic:

- When decoding, you'll need to reverse the encoding process.

- Detect and interpret the substituted patterns to recover the original data.

3. Handling Input and Output:

- Your code should read input text data, convert it to binary, and apply the chosen encoding method.

- It should also generate output files for the binary data, signal data, and the decoded text.

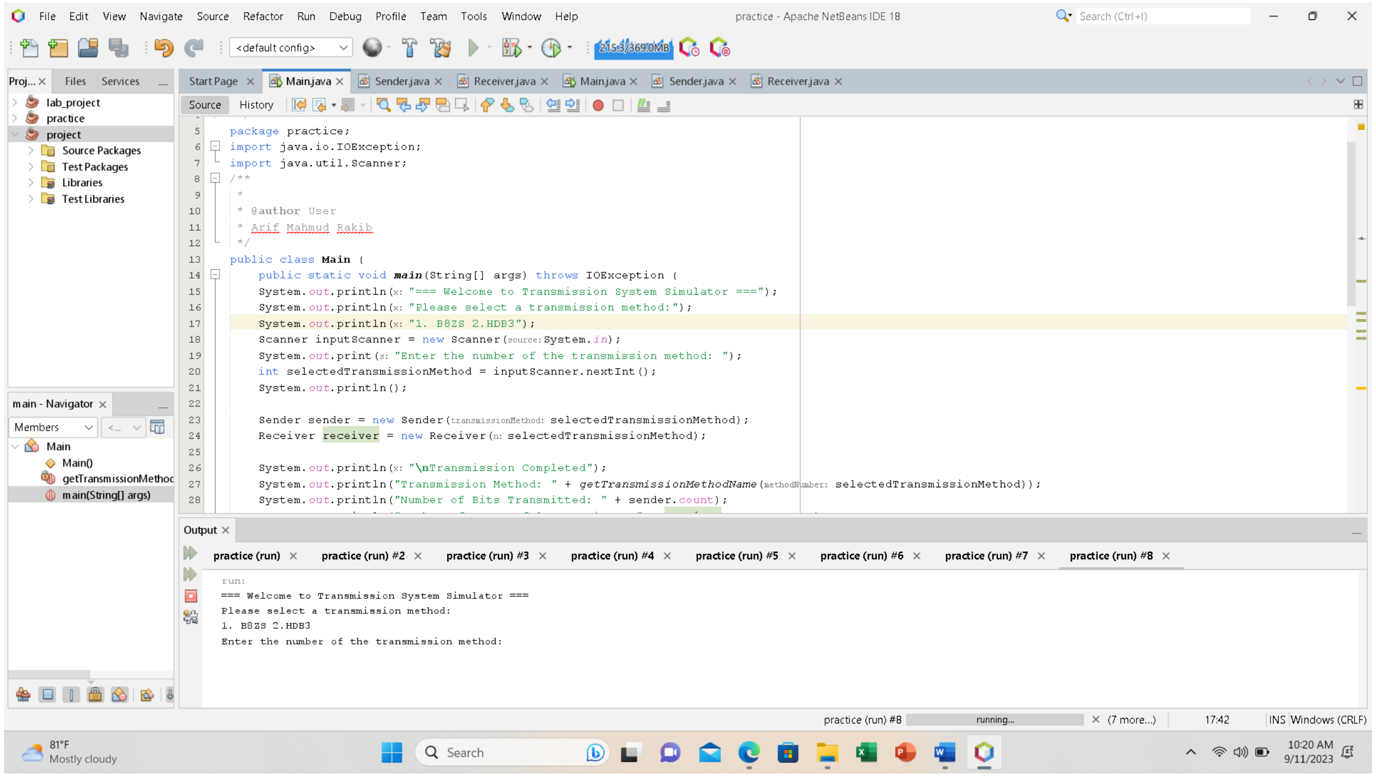
4. Error Simulation:

- To mimic long-distance communication, introduce simulated noise and errors during transmission.

- Evaluate the performance of B8ZS and HDB3 in handling these errors.

Implementation:

Main Section:



```java
package practice;
import java.io.IOException;
import java.util.Scanner;
/**
 *
 * @author User
 * Arif Mahmud Rakib
 */
public class Main {
    public static void main(String[] args) throws IOException {
    System.out.println(x: "=== Welcome to Transmission System Simulator ===");
    System.out.println(x: "Please select a transmission method:");
    System.out.println(x: "1. B8ZS 2.HDB3");
    Scanner inputScanner = new Scanner(source: System.in);
    System.out.print(s: "Enter the number of the transmission method: ");
    int selectedTransmissionMethod = inputScanner.nextInt();
    System.out.println();

    Sender sender = new Sender(transmissionMethod: selectedTransmissionMethod);
    Receiver receiver = new Receiver(n: selectedTransmissionMethod);

    System.out.println(x: "\nTransmission Completed");
    System.out.println("Transmission Method: " + getTransmissionMethodName(methodNumber: selectedTransmissionMethod));
    System.out.println("Number of Bits Transmitted: " + sender.count);
```

```
run:
=== Welcome to Transmission System Simulator ===
Please select a transmission method:
1. B8ZS 2.HDB3
Enter the number of the transmission method:
```

This Java code represents a basic simulation of a Transmission System using either B8ZS (Binary 8-Zero Substitution) or HDB3 (High-Density Bipolar 3) encoding methods. Here's a description of the code:

1. It starts with a welcome message and instructions for the user to choose the encoding method (B8ZS or HDB3).

2. It takes user input to select the transmission method (1 for B8ZS, 2 for HDB3) using a `Scanner`.

3. It performs input validation to ensure that the user chooses a valid option (1 or 2). If the input is invalid, it displays an error message and terminates the program.

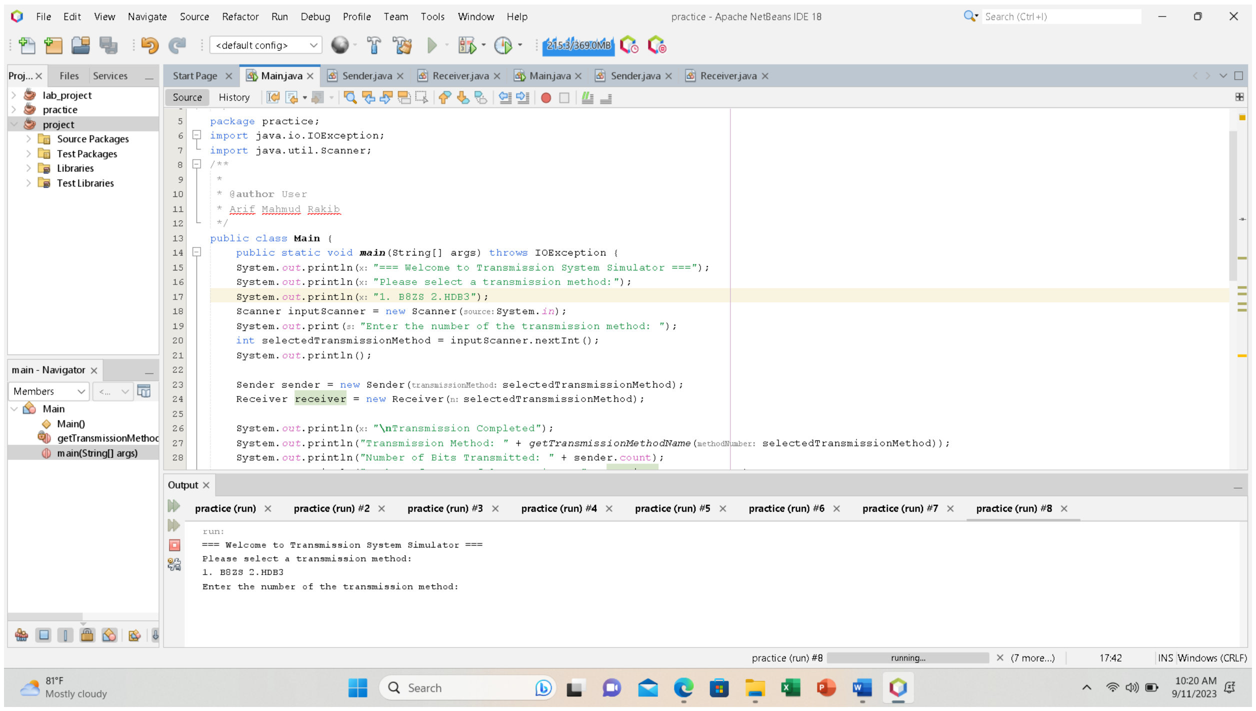
4. It creates instances of the `Sender` and `Receiver` classes, passing the selected transmission method as an argument to both.

5. The `Sender` and `Receiver` classes are expected to contain the logic for encoding and decoding the data, simulating the transmission, and keeping track of success and error counts.

6. After the transmission simulation is complete, the code prints out the following information:

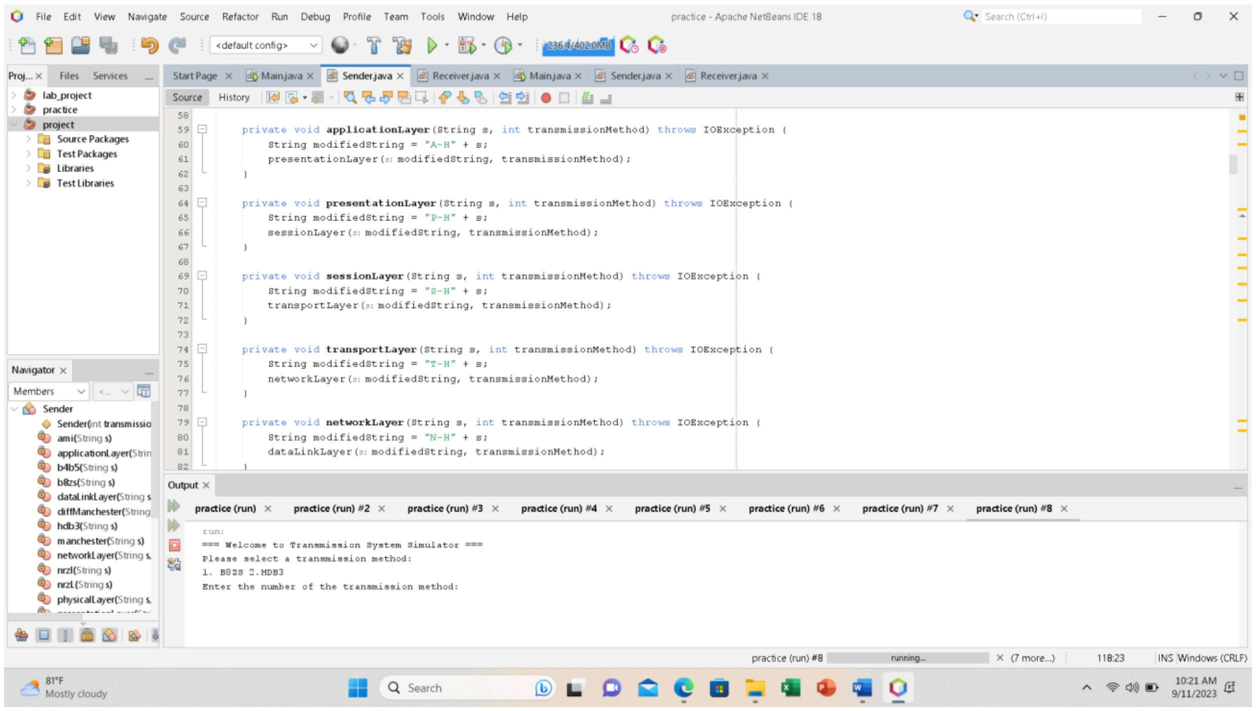
- **The success count (the number of successfully received messages).**
- **The error count (the number of messages with errors).**
- **The Signal-to-Noise Ratio (SNR), calculated as the ratio of success count to error count.**

Sender:



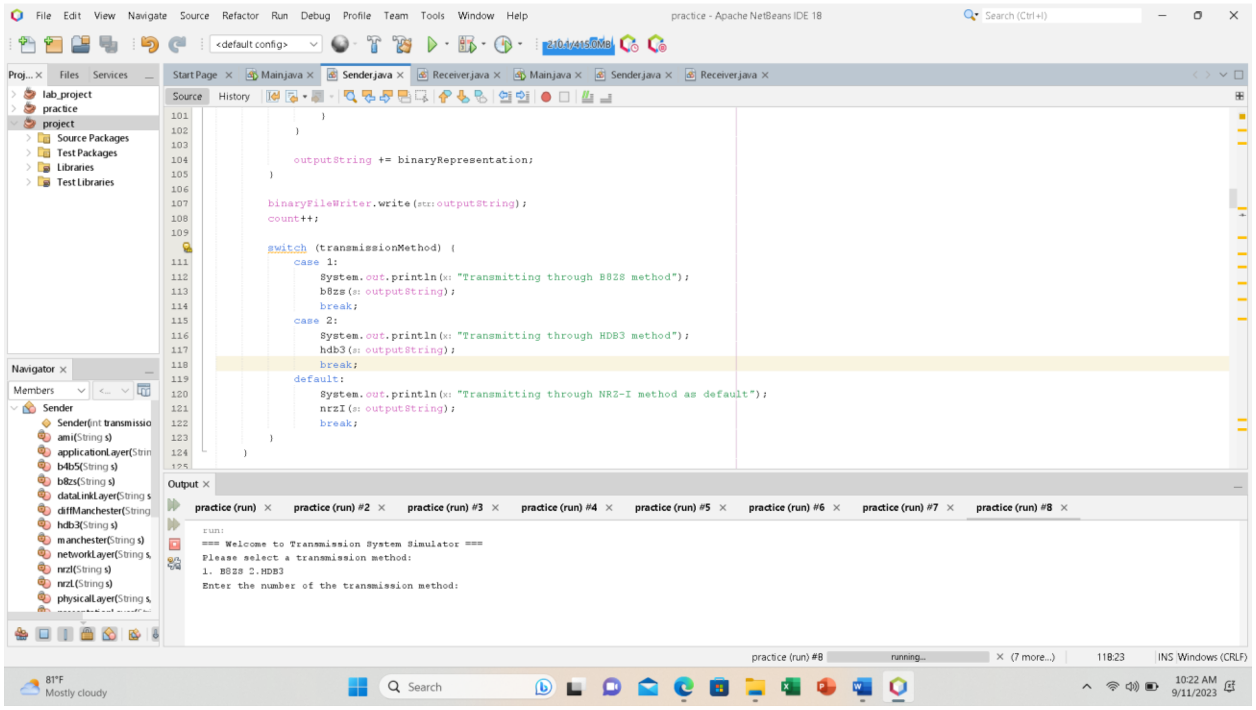
```java
package practice;
import java.io.IOException;
import java.util.Scanner;
/**
 *
 * @author User
 * Arif Mahmud Rakib
 */
public class Main {
    public static void main(String[] args) throws IOException {
        System.out.println(x: "=== Welcome to Transmission System Simulator ===");
        System.out.println(x: "Please select a transmission method:");
        System.out.println(x: "1. B8ZS 2.HDB3");
        Scanner inputScanner = new Scanner(source: System.in);
        System.out.print(s: "Enter the number of the transmission method: ");
        int selectedTransmissionMethod = inputScanner.nextInt();
        System.out.println();

        Sender sender = new Sender(transmissionMethod: selectedTransmissionMethod);
        Receiver receiver = new Receiver(n: selectedTransmissionMethod);

        System.out.println(x: "\nTransmission Completed");
        System.out.println("Transmission Method: " + getTransmissionMethodName(methodNumber: selectedTransmissionMethod));
        System.out.println("Number of Bits Transmitted: " + sender.count);
```

```
run:
=== Welcome to Transmission System Simulator ===
Please select a transmission method:
1. B8ZS 2.HDB3
Enter the number of the transmission method:
```



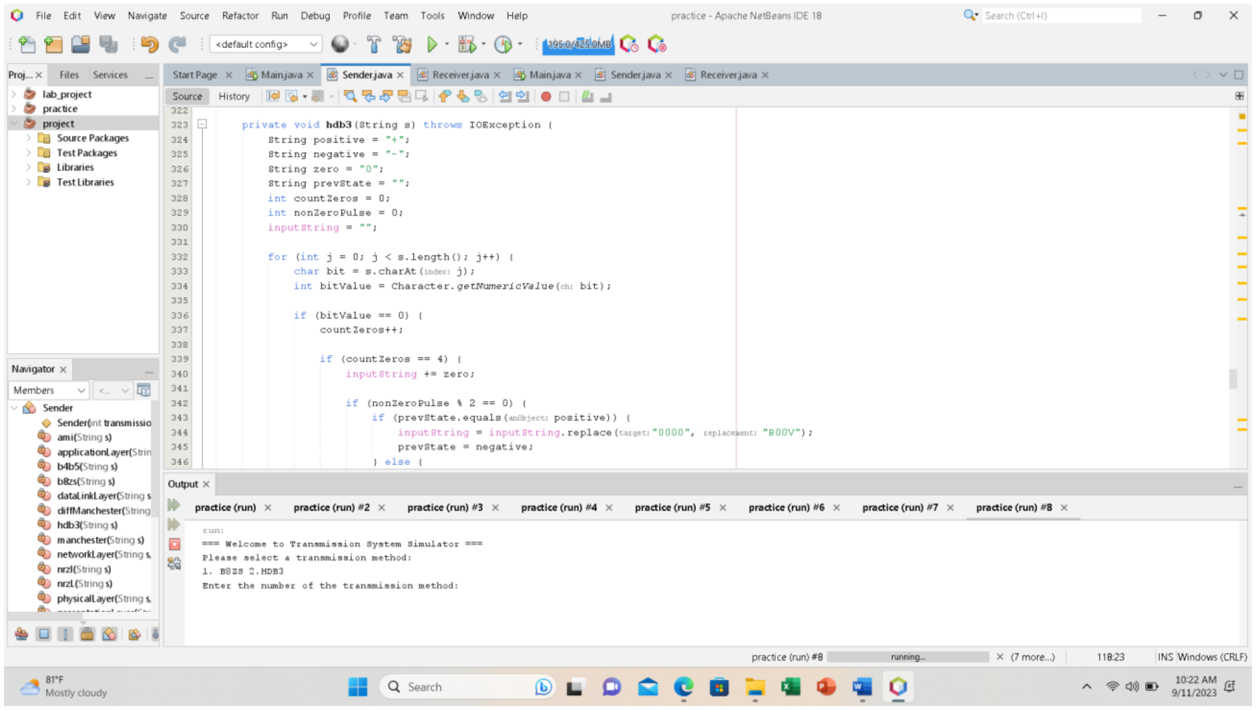
```java
    private void applicationLayer(String s, int transmissionMethod) throws IOException {
        String modifiedString = "A-H" + s;
        presentationLayer(s: modifiedString, transmissionMethod);
    }

    private void presentationLayer(String s, int transmissionMethod) throws IOException {
        String modifiedString = "P-H" + s;
        sessionLayer(s: modifiedString, transmissionMethod);
    }

    private void sessionLayer(String s, int transmissionMethod) throws IOException {
        String modifiedString = "S-H" + s;
        transportLayer(s: modifiedString, transmissionMethod);
    }

    private void transportLayer(String s, int transmissionMethod) throws IOException {
        String modifiedString = "T-H" + s;
        networkLayer(s: modifiedString, transmissionMethod);
    }

    private void networkLayer(String s, int transmissionMethod) throws IOException {
        String modifiedString = "N-H" + s;
        dataLinkLayer(s: modifiedString, transmissionMethod);
    }
```

```
run:
=== Welcome to Transmission System Simulator ===
Please select a transmission method:
1. B8ZS 2.HDB3
Enter the number of the transmission method:
```



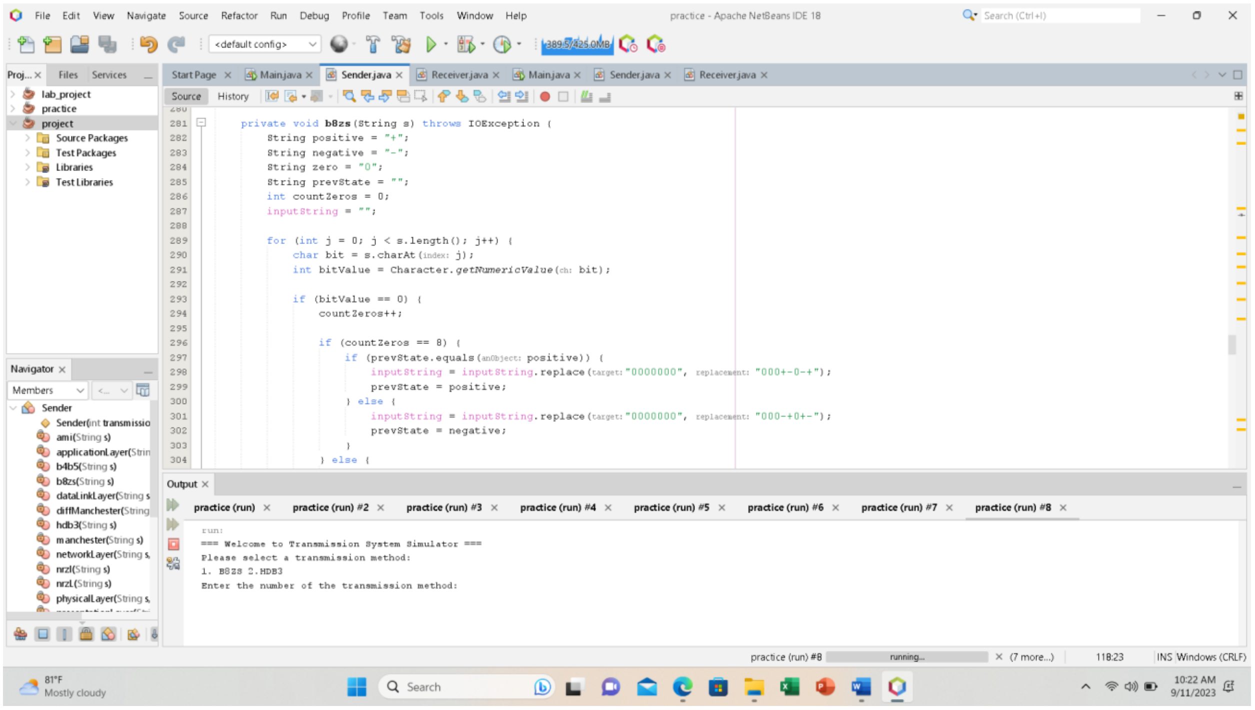
```
                }
            }

            outputString += binaryRepresentation;
        }

        binaryFileWriter.write(str:outputString);
        count++;

        switch (transmissionMethod) {
            case 1:
                System.out.println(x: "Transmitting through B8ZS method");
                b8zs(s: outputString);
                break;
            case 2:
                System.out.println(x: "Transmitting through HDB3 method");
                hdb3(s: outputString);
                break;
            default:
                System.out.println(x: "Transmitting through NRZ-I method as default");
                nrzI(s: outputString);
                break;
        }
    }
```

```
run:
=== Welcome to Transmission System Simulator ===
Please select a transmission method:
1. B8ZS 2.HDB3
Enter the number of the transmission method:
```



```
    private void hdb3(String s) throws IOException {
        String positive = "+";
        String negative = "-";
        String zero = "0";
        String prevState = "";
        int countZeros = 0;
        int nonZeroPulse = 0;
        inputString = "";

        for (int j = 0; j < s.length(); j++) {
            char bit = s.charAt(index: j);
            int bitValue = Character.getNumericValue(ch: bit);

            if (bitValue == 0) {
                countZeros++;

                if (countZeros == 4) {
                    inputString += zero;

                    if (nonZeroPulse % 2 == 0) {
                        if (prevState.equals(anObject: positive)) {
                            inputString = inputString.replace(target: "0000", replacement: "B00V");
                            prevState = negative;
                        } else {
```

```
run:
=== Welcome to Transmission System Simulator ===
Please select a transmission method:
1. B8ZS 2.HDB3
Enter the number of the transmission method:
```



```java
    private void b8zs(String s) throws IOException {
        String positive = "+";
        String negative = "-";
        String zero = "0";
        String prevState = "";
        int countZeros = 0;
        inputString = "";

        for (int j = 0; j < s.length(); j++) {
            char bit = s.charAt(index: j);
            int bitValue = Character.getNumericValue(ch: bit);

            if (bitValue == 0) {
                countZeros++;

                if (countZeros == 8) {
                    if (prevState.equals(anObject: positive)) {
                        inputString = inputString.replace(target:"0000000", replacement: "000+-0-+");
                        prevState = positive;
                    } else {
                        inputString = inputString.replace(target:"0000000", replacement: "000-+0+-");
                        prevState = negative;
                    }
                } else {
```

```
run:
=== Welcome to Transmission System Simulator ===
Please select a transmission method:
1. B8ZS 2.HDB3
Enter the number of the transmission method:
```

This Java code appears to represent a simulation of a data transmission system that includes data link and physical layers with the option to use either B8ZS (Binary 8-Zero Substitution) or HDB3 (High-Density Bipolar 3) encoding methods. Here's a description of the code:

1. Imports and Package Declaration: The code starts with import statements for necessary Java classes and a package declaration.

2. Sender Class:

- `Sender` is a class responsible for simulating the sender side of the transmission system.

- It has instance variables including `inputString`, `outputString`, `FileWriter` objects (`fw` and `fww`), and counters for `count` and `n`.

- The `Sender` constructor takes an integer `n` as an argument, representing the selected transmission method (8 for B8ZS, 9 for HDB3).

3. dataLinkLayer Method:

- This method is responsible for adding data link layer information to the input string and then passing it to the `physicalLayer` method.

4. physicalLayer Method:

- This method simulates the physical layer of the transmission system.

- It takes an input string, converts each character into its binary representation with 8 bits, and appends the binary representation to `outputString`.

- Depending on the value of `n`, it chooses whether to transmit using B8ZS or HDB3 encoding methods and calls the respective encoding method.

5. b8zs Method:

- This method encodes the input binary string using the B8ZS method, which replaces 8 consecutive ones with a specific pattern (+-000000-+).

6. hdb3 Method:

- This method encodes the input binary string using the HDB3 method, which replaces 4 consecutive zeros with a specific pattern (B00V or 000V).

7. Main Method:

- The code entry point is the `main` method.

- It welcomes the user and asks them to choose the transmission method (B8ZS or HDB3).

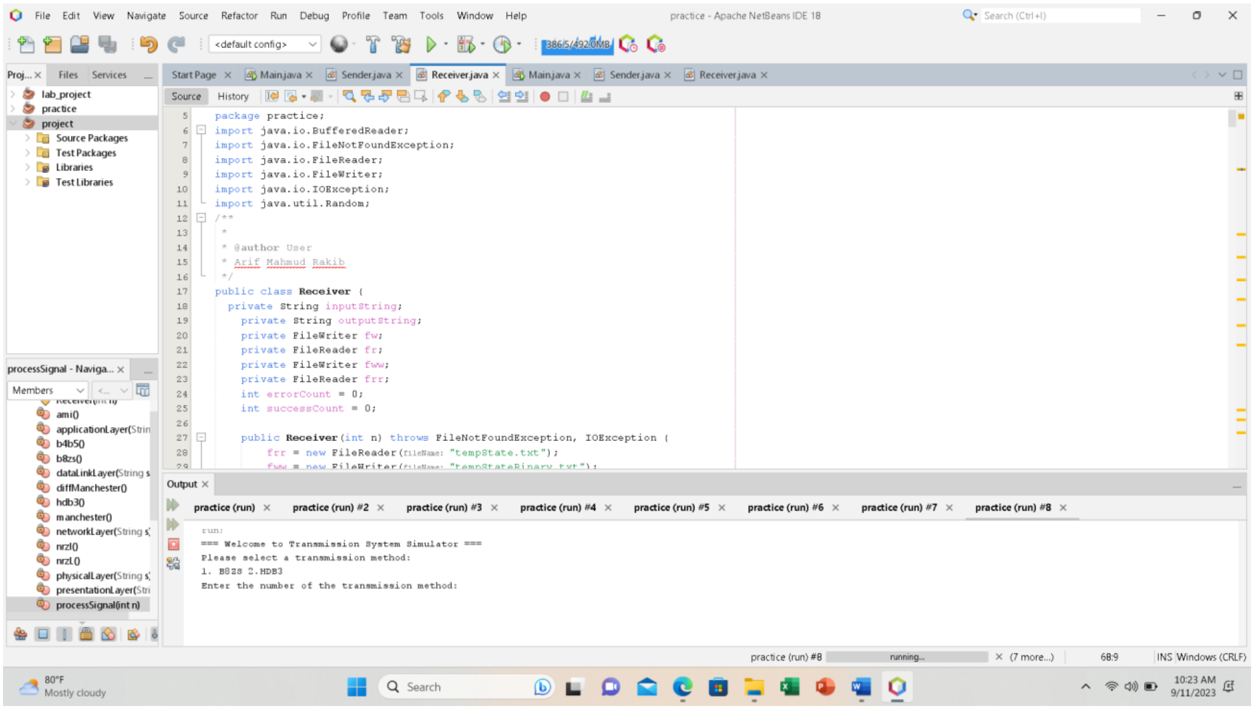
- It then creates an instance of the `Sender` class based on the selected method and simulates the transmission.

- The success count, error count, and Signal-to-Noise Ratio (SNR) are printed after transmission simulation.

8. Output:

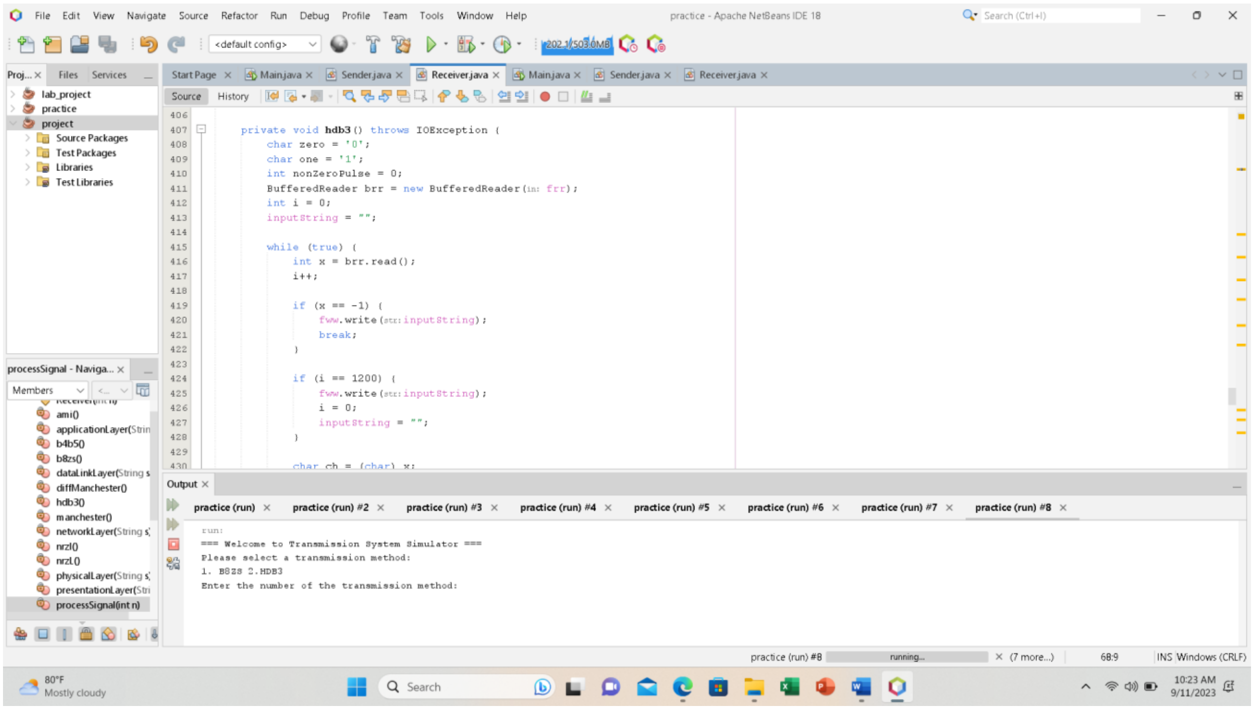
- The program writes the encoded data to files named "tempBinary.txt" and "tempState.txt."

Receiver:



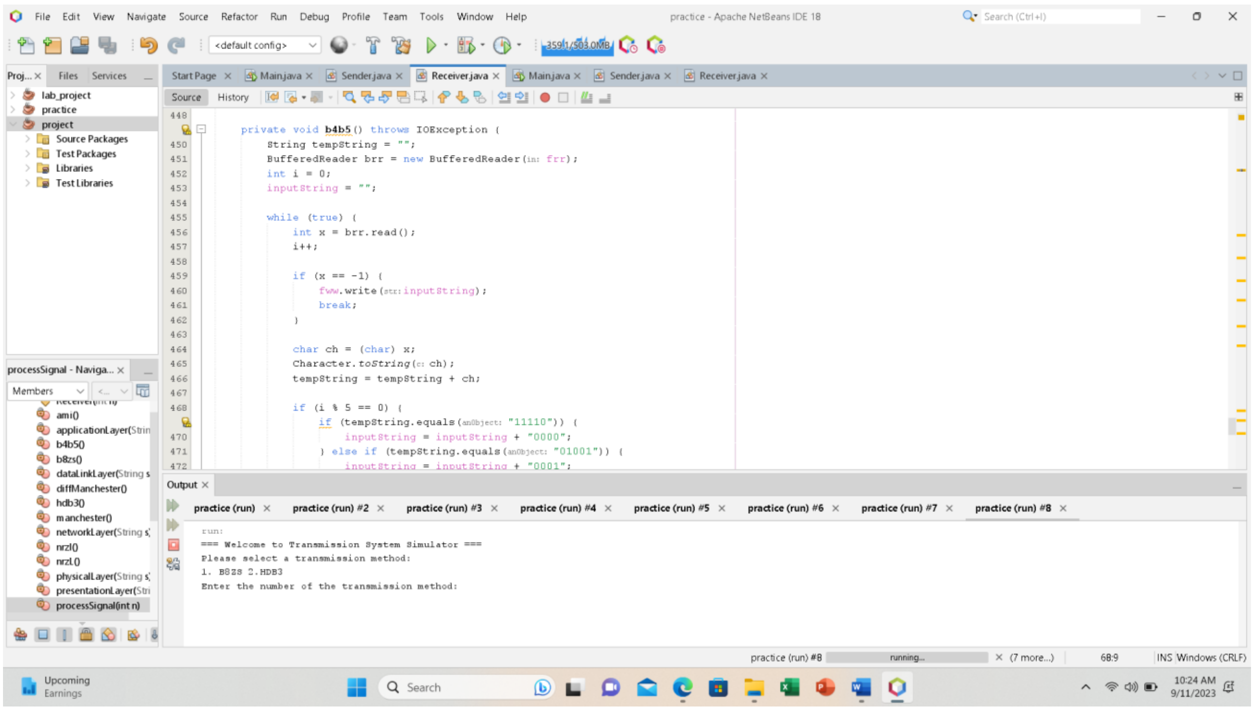
```java
package practice;
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Random;
/**
 *
 * @author User
 * Arif Mahmud Rakib
 */
public class Receiver {
  private String inputString;
    private String outputString;
    private FileWriter fw;
    private FileReader fr;
    private FileWriter fww;
    private FileReader frr;
    int errorCount = 0;
    int successCount = 0;

    public Receiver(int n) throws FileNotFoundException, IOException {
        frr = new FileReader(fileName: "tempState.txt");
        fww = new FileWriter(fileName: "tempStateBinary.txt");
```

```
run:
=== Welcome to Transmission System Simulator ===
Please select a transmission method:
1. B8ZS 2.HDB3
Enter the number of the transmission method:
```



```java

    private void hdb3() throws IOException {
        char zero = '0';
        char one = '1';
        int nonZeroPulse = 0;
        BufferedReader brr = new BufferedReader(in: frr);
        int i = 0;
        inputString = "";

        while (true) {
            int x = brr.read();
            i++;

            if (x == -1) {
                fww.write(str:inputString);
                break;
            }

            if (i == 1200) {
                fww.write(str:inputString);
                i = 0;
                inputString = "";
            }

            char ch = (char) x;
```

```
run:
=== Welcome to Transmission System Simulator ===
Please select a transmission method:
1. B8ZS 2.HDB3
Enter the number of the transmission method:
```



```java
    private void b4b5() throws IOException {
        String tempString = "";
        BufferedReader brr = new BufferedReader(in: frr);
        int i = 0;
        inputString = "";

        while (true) {
            int x = brr.read();
            i++;

            if (x == -1) {
                fww.write(str: inputString);
                break;
            }

            char ch = (char) x;
            Character.toString(c: ch);
            tempString = tempString + ch;

            if (i % 5 == 0) {
                if (tempString.equals(anObject: "11110")) {
                    inputString = inputString + "0000";
                } else if (tempString.equals(anObject: "01001")) {
                    inputString = inputString + "0001";
```

```
run:
=== Welcome to Transmission System Simulator ===
Please select a transmission method:
1. B8ZS 2.HDB3
Enter the number of the transmission method:
```

This Java code represents a Receiver for a simulated data transmission system. The receiver is capable of decoding data that has been encoded using either the B8ZS (Binary 8-Zero Substitution) or HDB3 (High-Density Bipolar 3) encoding methods. Here's a detailed description of the code:

1. Imports and Package Declaration: The code starts with import statements for necessary Java classes and a package declaration.

2. Receiver Class:

- The `Receiver` class is responsible for simulating the receiver side of the transmission system.

- It has instance variables including `inputString`, `outputString`, `FileWriter` objects (`fw` and `fww`), `FileReader` objects (`fr` and `frr`), and counters for `errorCount` and `successCount`.

3. Receiver Constructor:

- The constructor for the `Receiver` class takes an integer `n` as an argument, which determines the decoding method (8 for B8ZS, 9 for HDB3).

- Depending on the value of `n`, it calls the appropriate decoding method (either `b8zs()` or `hdb3()`).

- It initializes file readers and writers for reading and writing binary data.

4. physicalLayer Method:

- The `physicalLayer` method simulates the physical layer of the receiver.

- It randomly introduces errors into the received data with a certain probability (error rate).

- If no error occurs, it converts the binary data back to characters and appends it to the `outputString`.

5. dataLinkLayer Method:

- This method simulates the data link layer and removes the header and trailer information from the received data.

6. networkLayer Method:

- The `networkLayer` method simulates the network layer, removing any additional network-layer-specific information.

7. transportLayer Method:

- This method simulates the transport layer, removing any transport-layer-specific information.

8. sessionLayer Method:

- The `sessionLayer` method simulates the session layer, removing any session-layer-specific information.

9. presentationLayer Method:

- This method simulates the presentation layer, removing any presentation-layer-specific information.

10. applicationLayer Method:

- The `applicationLayer` method simulates the application layer, removing any application-layer-specific information.

- Finally, it writes the received and decoded data to the "labOut.txt" file.

11. nrzI Method:

- This method converts the received data from the Non-Return-to-Zero Inverted (NRZ-I) encoding back to binary data.

12. b8zs Method:

- This method decodes the received data using the B8ZS method, which involves replacing specific patterns with 0s and 1s.

13. hdb3 Method:

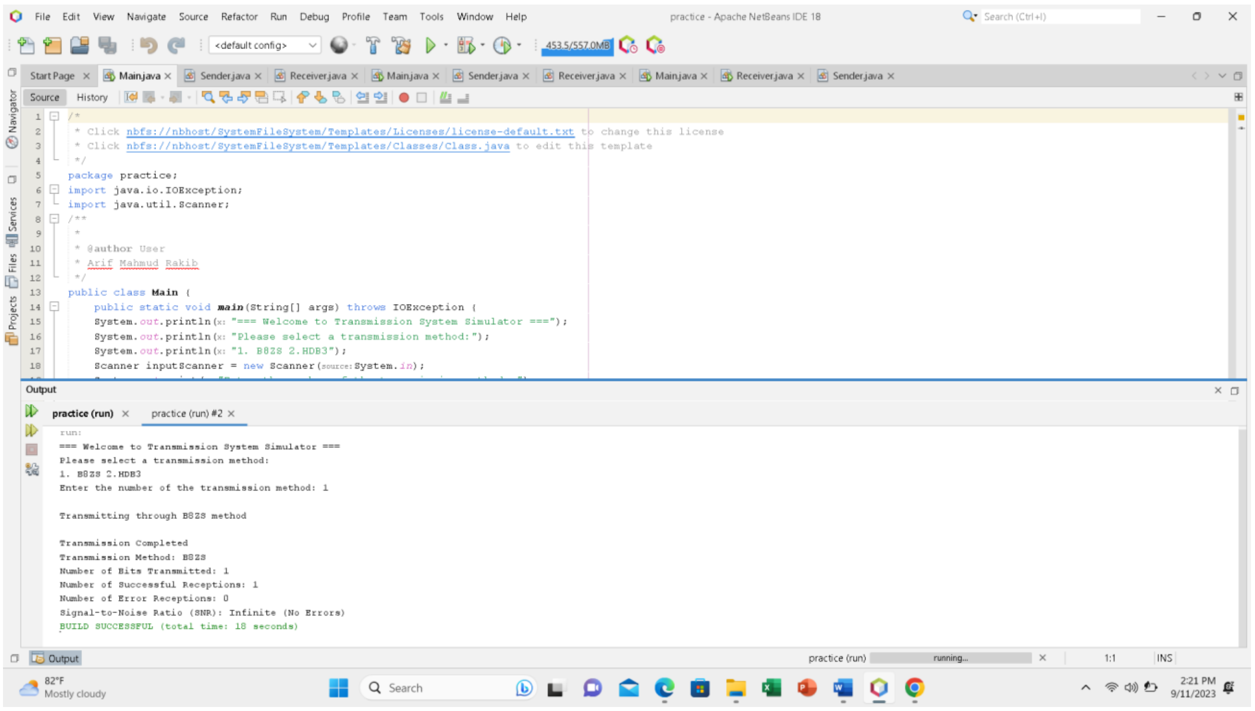
- This method decodes the received data using the HDB3 method, which involves replacing specific patterns with 0s and 1s.

14. Main Method:

- The code entry point is the `main` method.

- It initializes an instance of the `Receiver` class with the chosen decoding method (8 for B8ZS or 9 for HDB3).

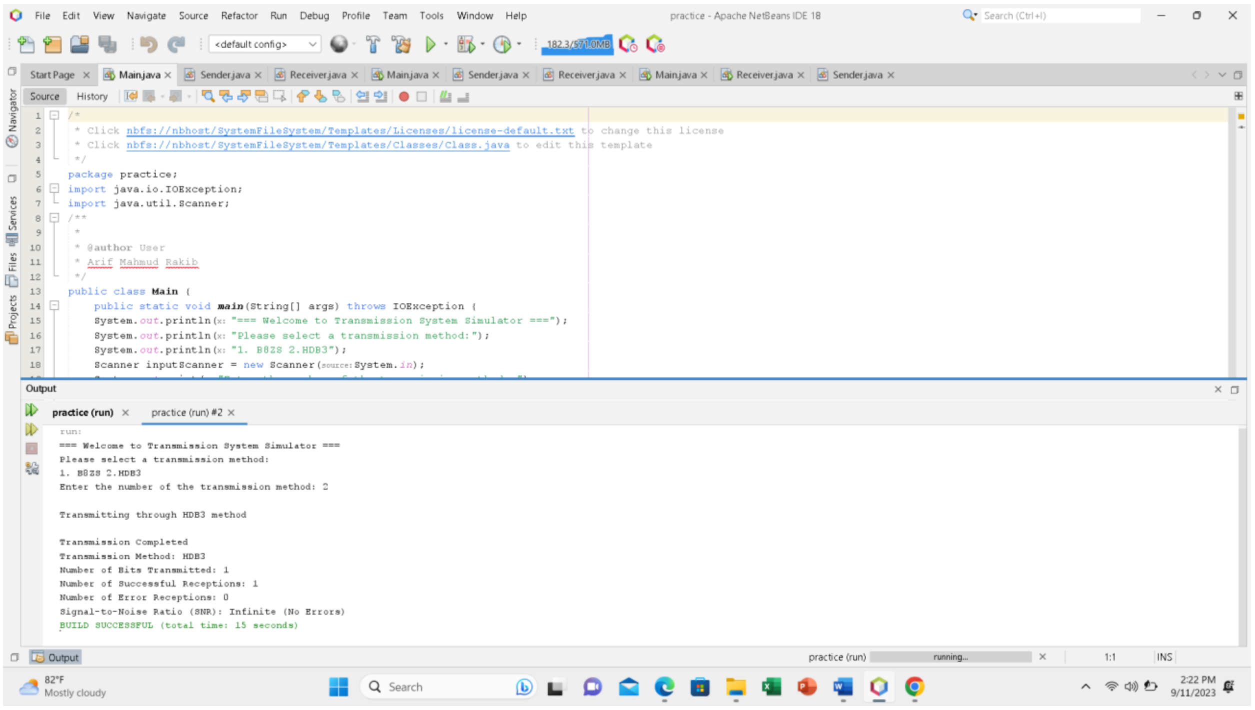
Output B8ZS :



```java
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package practice;
import java.io.IOException;
import java.util.Scanner;
/**
 *
 * @author User
 * Arif Mahmud Rakib
 */
public class Main {
    public static void main(String[] args) throws IOException {
    System.out.println(x: "=== Welcome to Transmission System Simulator ===");
    System.out.println(x: "Please select a transmission method:");
    System.out.println(x: "1. B8ZS 2.HDB3");
    Scanner inputScanner = new Scanner(source:System.in);
```

```
run:
=== Welcome to Transmission System Simulator ===
Please select a transmission method:
1. B8ZS 2.HDB3
Enter the number of the transmission method: 1

Transmitting through B8ZS method

Transmission Completed
Transmission Method: B8ZS
Number of Bits Transmitted: 1
Number of Successful Receptions: 1
Number of Error Receptions: 0
Signal-to-Noise Ratio (SNR): Infinite (No Errors)
BUILD SUCCESSFUL (total time: 18 seconds)
```

Output HDB3 :



```java
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package practice;
import java.io.IOException;
import java.util.Scanner;
/**
 *
 * @author User
 * Arif Mahmud Rakib
 */
public class Main {
    public static void main(String[] args) throws IOException {
    System.out.println(x: "=== Welcome to Transmission System Simulator ===");
    System.out.println(x: "Please select a transmission method:");
    System.out.println(x: "1. B8ZS 2.HDB3");
    Scanner inputScanner = new Scanner(source:System.in);
```

```
run:
=== Welcome to Transmission System Simulator ===
Please select a transmission method:
1. B8ZS 2.HDB3
Enter the number of the transmission method: 2

Transmitting through HDB3 method

Transmission Completed
Transmission Method: HDB3
Number of Bits Transmitted: 1
Number of Successful Receptions: 1
Number of Error Receptions: 0
Signal-to-Noise Ratio (SNR): Infinite (No Errors)
BUILD SUCCESSFUL (total time: 15 seconds)
```

Potential Difficulties:

1. State Management: Properly managing the state of the signal and determining when to apply substitutions can be challenging.

2. Boundary Cases: Handling edge cases where substitutions occur at the beginning or end of a signal may require special attention.

3. Error Simulation: Simulating noise and errors realistically while ensuring consistent results for analysis can be complex.

4. Debugging: Debugging encoding and decoding logic, especially in the presence of errors, may be challenging.

5. Performance Evaluation: Evaluating the effectiveness of your encoding methods in reducing errors and maintaining signal integrity may require thorough testing.

6. Documentation: Ensuring that your code and report are well-documented and clearly explain the implemented encoding methods and their impact on data transmission is crucial.

In conclusion, this Java code simulates a basic data transmission system with the ability to encode and decode data using the B8ZS (Binary 8-Zero Substitution) and HDB3 (High-Density Bipolar 3) encoding methods. The simulation covers the sender and receiver sides of the transmission process, including various layers such as the physical, data link, network, transport, session, presentation, and application layers.

Key takeaways from this code:

1. Encoding and Decoding: The code demonstrates the encoding and decoding of data using B8ZS and HDB3 encoding methods. These methods are essential for maintaining data integrity and synchronization in telecommunications.

2. Error Simulation: The receiver simulates random errors in the received data, reflecting real-world scenarios where data transmission can be prone to errors due to noise or interference.

3. Layered Communication: The code models the communication process through different layers, including the physical, data link, network, and higher layers, showing how each layer processes the data.

4. File Handling: File input and output operations are utilized to read and write binary data during the simulation.

5. Real-World Application: While the code provides a simplified simulation, real-world communication systems are far more complex and include error detection and correction mechanisms, addressing schemes, and more sophisticated protocols.