**Ex No: 6**

**Date:**

**IMPLEMENTATION OF SLIDING WINDOW PROTOCOL**

**Aim:**

Assume a network scenario where N is the sender’s window size = ‘m’ & damaged frame is ‘x’;

* 1. At this situation, the sender sends frame 1 to ‘m’ before receiving the knowledge of frame 1. All the frames are numbered to deal with the most and duplicate frames. If the sender does not receive the receiver’s acknowledgement, then all the frames available in the current window ie., 1 to ‘m’ will be retransmitted.

(b) The sender will resend only the damaged frame ie., frame ‘x’

**(a) Go- Back-N Protocol**

**Theory:**

**Go-Back-N ARQ** uses the concept of protocol pipelining, i.e. the sender can send multiple frames before receiving the acknowledgement for the first frame. There is a finite number of frames, and the frames are numbered sequentially. The number of frames that can be sent depends on the sender's window size.

**Go-Back-N ARQ** is a specific instance of the [automatic repeat request](https://en.wikipedia.org/wiki/Automatic_repeat_request) (ARQ) protocol, in which the sending process continues to send a number of [frames](https://en.wikipedia.org/wiki/Data_frame) specified by a *window size* even without receiving an [acknowledgement](https://en.wikipedia.org/wiki/Acknowledgement_%28data_networks%29) (ACK) packet from the receiver. It is a special case of the general [sliding window protocol](https://en.wikipedia.org/wiki/Sliding_window_protocol) with the transmit window size of N and receive window size of even 1. It can transmit N frames to the peer before requiring an ACK.

**Algorithm:**

**SENDER:**

Assuming Total no. of frames to be transmitted to be **‘nf’,** with a window size of **’sws’**

**Initialize Server:**

* Create a ServerSocket to listen for connections on a specific port (port 10 in this case).
* Accept a connection from the client (Socket s).

**Input Setup:**

* Set up BufferedReader objects to read inputs from the console (user) and from the client.
* Set up an array sbuff[] to store the frames to be sent.

**Window Size and Frame Information:**

* Ask the user to input the sliding window size (sws).
* Ask the user to input the total number of frames to be sent (nf).
* Send the total number of frames (nf) to the receiver.

**Frame Data Input:**

* Ask the user to input the actual messages to send for each frame and store them in sbuff[].

**Sliding Window Transmission:**

* Initialize base (the starting frame of the window) to 0 and nextFrame to 0 (the next frame to be sent).
* While not all frames are acknowledged (base < nf):
  + **Send Frames:**
    - Send all frames within the sliding window where nextFrame - base < sws.
    - For each frame, print the frame number and message, and send it to the receiver.
  + **Wait for Acknowledgment:**
    - After sending the frames, wait for an acknowledgment from the receiver.
    - Implement a timeout of 10 seconds to detect if the acknowledgment is received within the time frame.
  + **Acknowledgment Handling:**
    - If an acknowledgment is received for a frame (ano), move the window base to ano + 1.
    - If all frames are acknowledged, terminate the process.
    - If no acknowledgment is received within the timeout, retransmit all frames starting from base.

**Frame Resending (Timeout Handling):**

* If the acknowledgment is not received within the timeout period, retransmit all the frames in the window starting from base.

**Termination:**

* Once all frames are acknowledged, print a message indicating that all frames have been successfully acknowledged.
* Close the socket connection.

**RECEIVER:**

Assuming the **damaged frame number to be 2.**

**Establish Connection:**

* Create a Socket to connect to the sender on the same machine (InetAddress.getLocalHost()) using port 10.

**Input and Output Setup:**

* Set up BufferedReader to receive data (frames) from the sender.
* Set up a PrintStream to send acknowledgment back to the sender.

**Initialization:**

* Initialize expectedFrame to 0, which represents the frame number the receiver is expecting.
* Create a buffer rbuf[] to store the received frames.
* Initialize a flag damagedFrameReceived to simulate a damaged frame (in this case, assume that frame 2 is damaged).

**Receive Total Number of Frames:**

* Receive the total number of frames (totalFrames) from the sender.
* Print the total number of frames expected.

**Receiving and Handling Frames:**

* While expectedFrame is less than totalFrames, keep receiving frames from the sender.
  + **Receive Frame:**
    - Read the incoming message from the sender and split it into two parts: the frame number (frameNumber) and the frame content (frameContent).
  + **Simulate Frame Damage:**
    - If the frame number is the predefined damaged frame (damagedFrame) and it hasn’t been previously received:
      * Simulate the damage by not sending an acknowledgment for the frame.
      * Skip the rest of the loop to avoid further processing of the damaged frame.
  + **Frame Handling:**
    - If the received frame is the expected frame (frameNumber == expectedFrame):
      * Store the frame content in the buffer rbuf[].
      * Print a message indicating the correct receipt of the frame.
      * Send an acknowledgment (expectedFrame) to the sender for the successfully received frame.
      * Increment the expectedFrame to move to the next frame.

**Termination:**

* Once all frames have been successfully received and acknowledged (expectedFrame >= totalFrames), print a message indicating that all frames have been received.
* Close the socket connection.

**Program:**

**SENDER:**

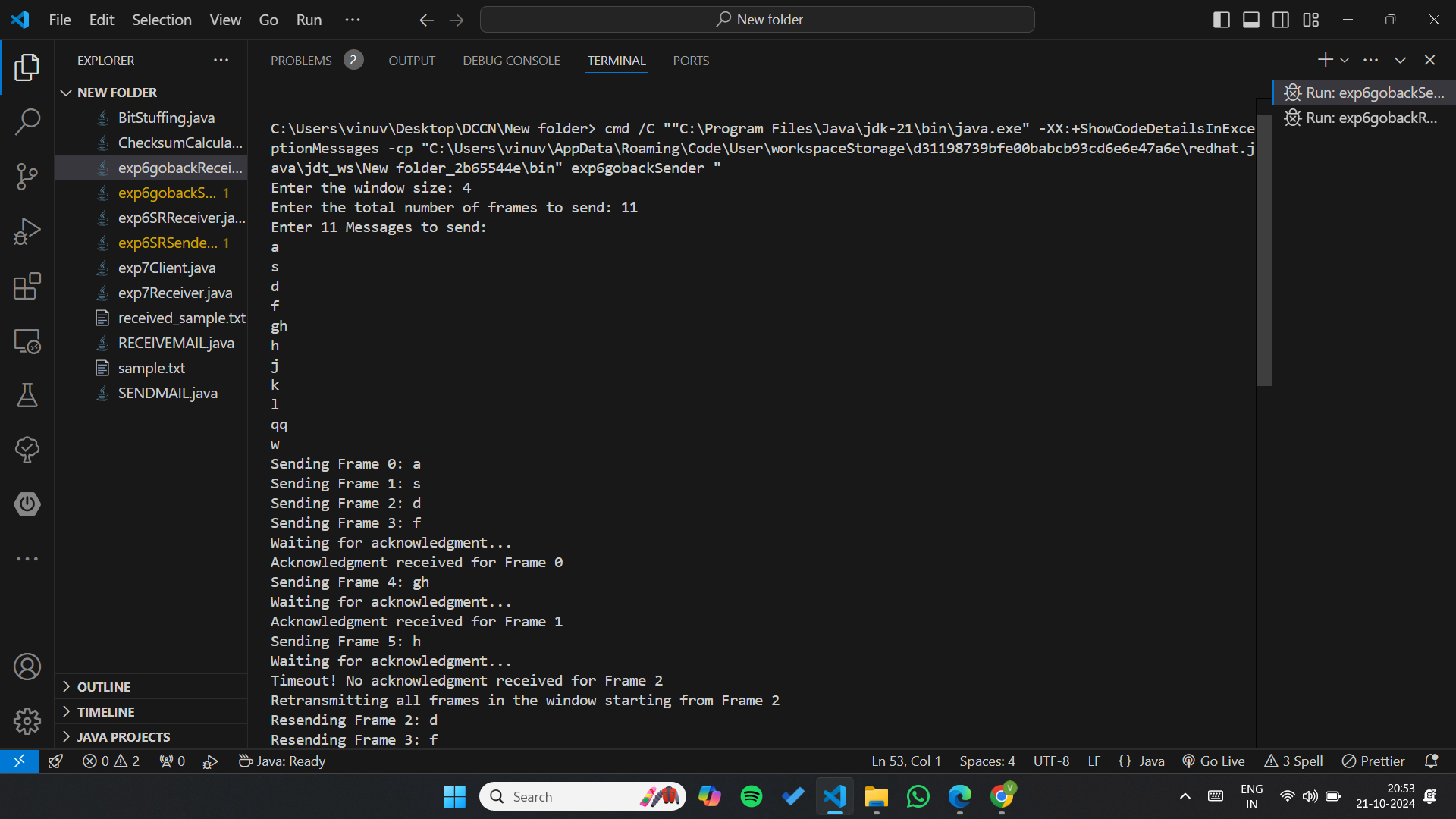
import java.net.\*;  
import java.io.\*;  
  
public class SLIDSENDER {  
 public static void main(String[] a) throws Exception {  
 ServerSocket ser = new ServerSocket(10);  
 Socket s = ser.accept();  
  
 BufferedReader in = new BufferedReader(new InputStreamReader(System.*in*));  
 BufferedReader in1 = new BufferedReader(new InputStreamReader(s.getInputStream()));  
 String sbuff[] = new String[100];  
 PrintStream p;  
  
 int sws, nf, ano, i, base = 0, nextFrame = 0;  
 boolean ackReceived = false;  
  
 // Get the window size input from the user  
 System.*out*.print("Enter the window size: ");  
 sws = Integer.*parseInt*(in.readLine());  
  
 // Get the total number of frames  
 System.*out*.print("Enter the total number of frames to send: ");  
 nf = Integer.*parseInt*(in.readLine());  
  
 // Inform the receiver about the total number of frames  
 p = new PrintStream(s.getOutputStream());  
 p.println(nf);  
  
 // Reading all frames from the user and storing them in the buffer  
 System.*out*.println("Enter " + nf + " Messages to send:");  
 for (i = 0; i < nf; i++) {  
 sbuff[i] = in.readLine();  
 }  
  
 do {  
 // Send frames within the window  
 while (nextFrame < nf && (nextFrame - base) < sws) {  
 p = new PrintStream(s.getOutputStream());  
 System.*out*.println("Sending Frame " + nextFrame + ": " + sbuff[nextFrame]);  
 p.println(nextFrame + ":" + sbuff[nextFrame]); // Send frame number and message  
 nextFrame++;  
 }  
  
 // Waiting for acknowledgment  
 System.*out*.println("Waiting for acknowledgment...");  
  
 long startTime = System.*currentTimeMillis*();  
 int timeout = 5000; // 5 seconds timeout  
  
 while (!ackReceived && (System.*currentTimeMillis*() - startTime) < timeout) {  
 // Check if acknowledgment is received  
 if (in1.ready()) {  
 ano = Integer.*parseInt*(in1.readLine());  
 System.*out*.println("Acknowledgment received for Frame " + ano);  
  
 // Slide the window base to the next unacknowledged frame  
 base = ano + 1;  
 ackReceived = true;  
  
 // If all frames are acknowledged, exit the loop  
 if (base >= nf) {  
 System.*out*.println("All frames acknowledged.");  
 break;  
 }  
 }  
 }  
  
 // If acknowledgment is not received, timeout occurred  
 if (!ackReceived) {  
 System.*out*.println("Timeout! No acknowledgment received for Frame " + base);  
 System.*out*.println("Retransmitting all frames in the window starting from Frame " + base);  
  
 // Resend all frames in the window starting from base  
 for (i = base; i < nextFrame; i++) {  
 p = new PrintStream(s.getOutputStream());  
 System.*out*.println("Resending Frame " + i + ": " + sbuff[i]);  
 p.println(i + ":" + sbuff[i]); // Send frame number and message  
 }  
 } else {  
 ackReceived = false;  
 }  
  
 } while (base < nf); // Continue until all frames are acknowledged  
  
 s.close();  
 }  
}

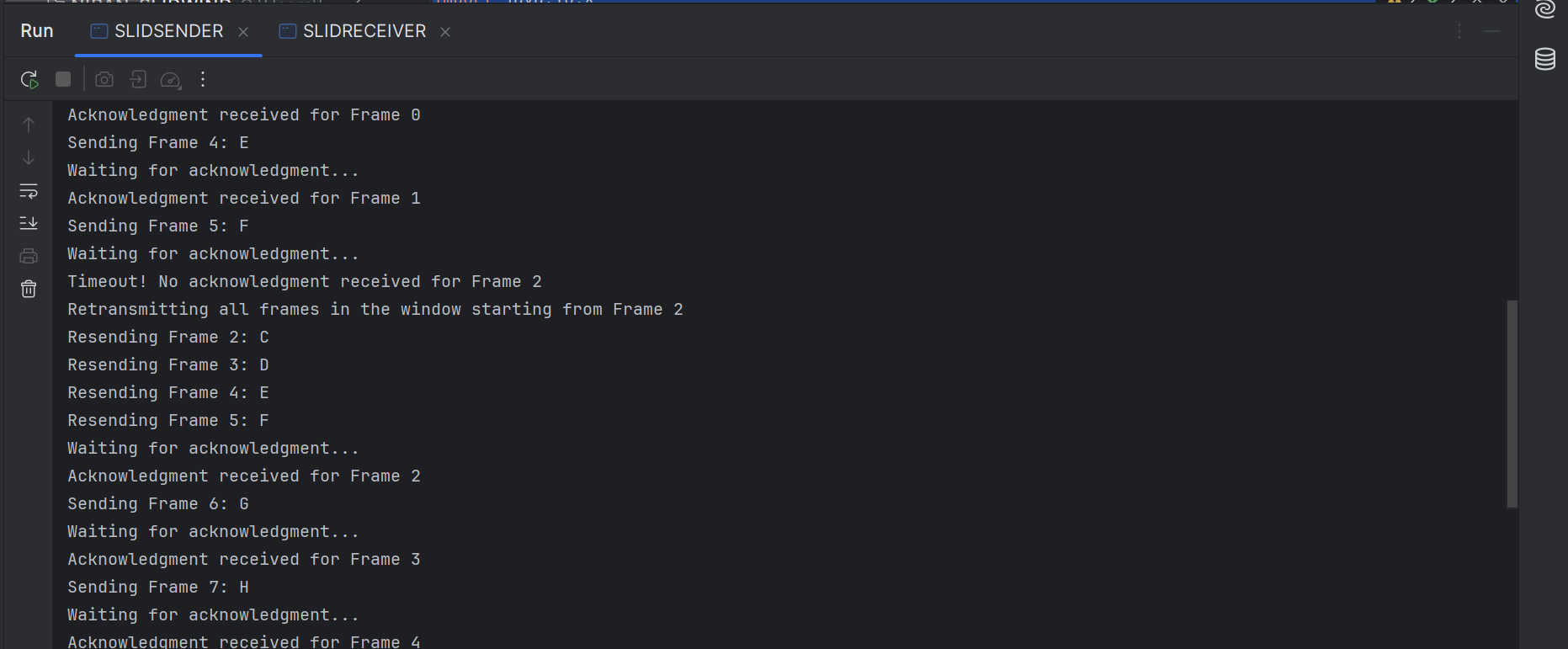
**RECEIVER:**

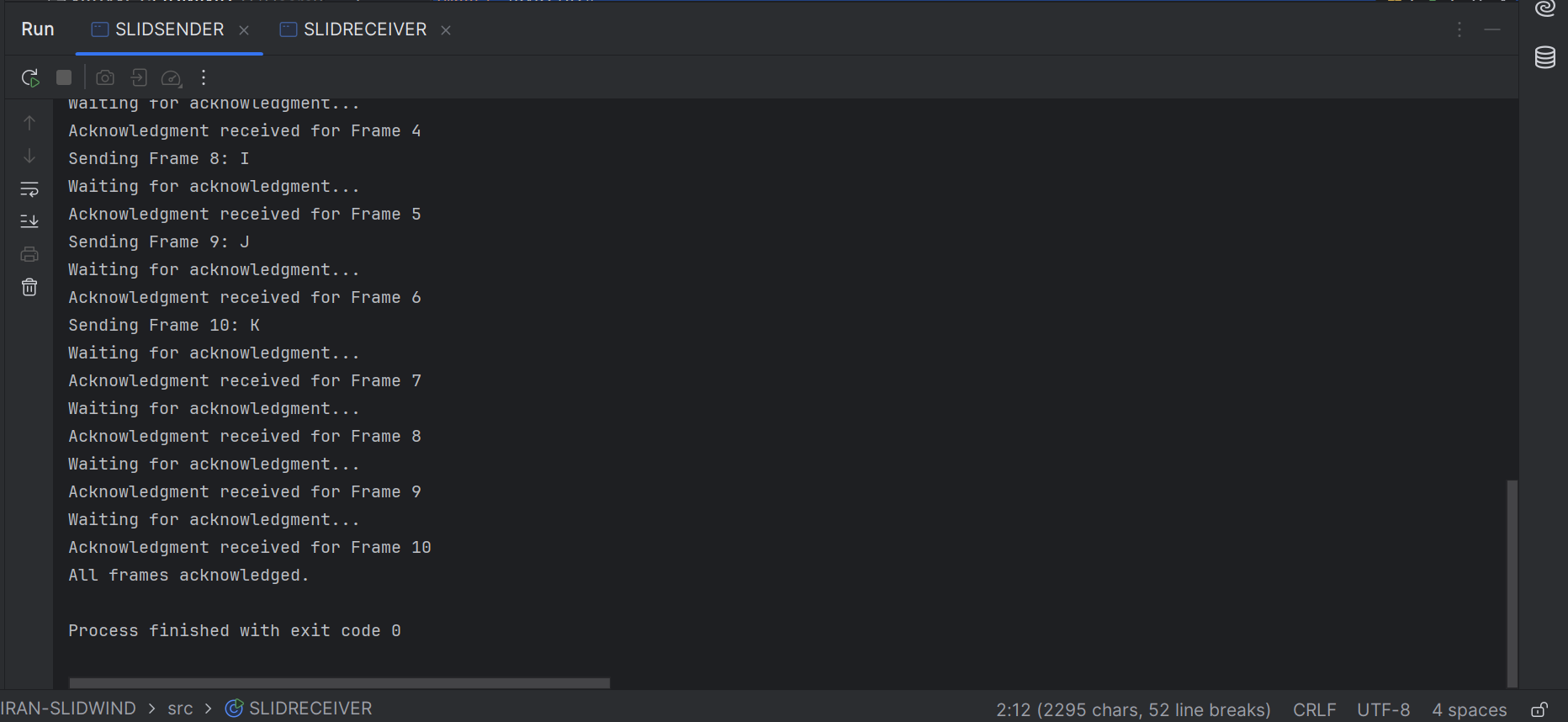
import java.net.\*;  
import java.io.\*;  
  
class SLIDRECEIVER {  
 public static void main(String[] a) throws Exception {  
 Socket s = new Socket(InetAddress.*getLocalHost*(), 10);  
 BufferedReader in = new BufferedReader(new InputStreamReader(s.getInputStream()));  
 PrintStream p = new PrintStream(s.getOutputStream());  
  
 int expectedFrame = 0, totalFrames;  
 String rbuf[] = new String[100];  
 int damagedFrame = 2; // Assume frame 2 is damaged on first attempt  
 boolean damagedFrameReceived = false; // Flag to track if the damaged frame has been received  
  
 System.*out*.println("Receiver is ready...");  
  
 // Read the total number of frames to be received (sent by the sender)  
 totalFrames = Integer.*parseInt*(in.readLine());  
 System.*out*.println("Total number of frames to receive: " + totalFrames);  
  
 while (expectedFrame < totalFrames) {  
 // Receive the frame from the sender  
 String frameMessage = in.readLine();  
  
 // Split the incoming message to get the frame number and content  
 String[] parts = frameMessage.split(":");  
 int frameNumber = Integer.*parseInt*(parts[0]);  
 String frameContent = parts[1];  
  
 // Simulate frame damage for the first attempt  
 if (frameNumber == damagedFrame && !damagedFrameReceived) {  
 System.*out*.println("Frame " + frameNumber + " is damaged. No acknowledgment sent.");  
 damagedFrameReceived = true; // Mark that the damaged frame has been received  
 continue; // Don't send an acknowledgment for the damaged frame  
 }  
  
 // If the frame is expected  
 if (frameNumber == expectedFrame) {  
 System.*out*.println("Received Frame " + expectedFrame + ": " + frameContent);  
 rbuf[expectedFrame] = frameContent;  
  
 // Send acknowledgment for the correctly received frame  
 System.*out*.println("Sending acknowledgment for Frame " + expectedFrame);  
 p.println(expectedFrame);  
 expectedFrame++;  
 }  
 }  
  
 System.*out*.println("All frames received and acknowledged.");  
 s.close(); // Close the socket when done  
 }  
}

**Screenshot of output:**

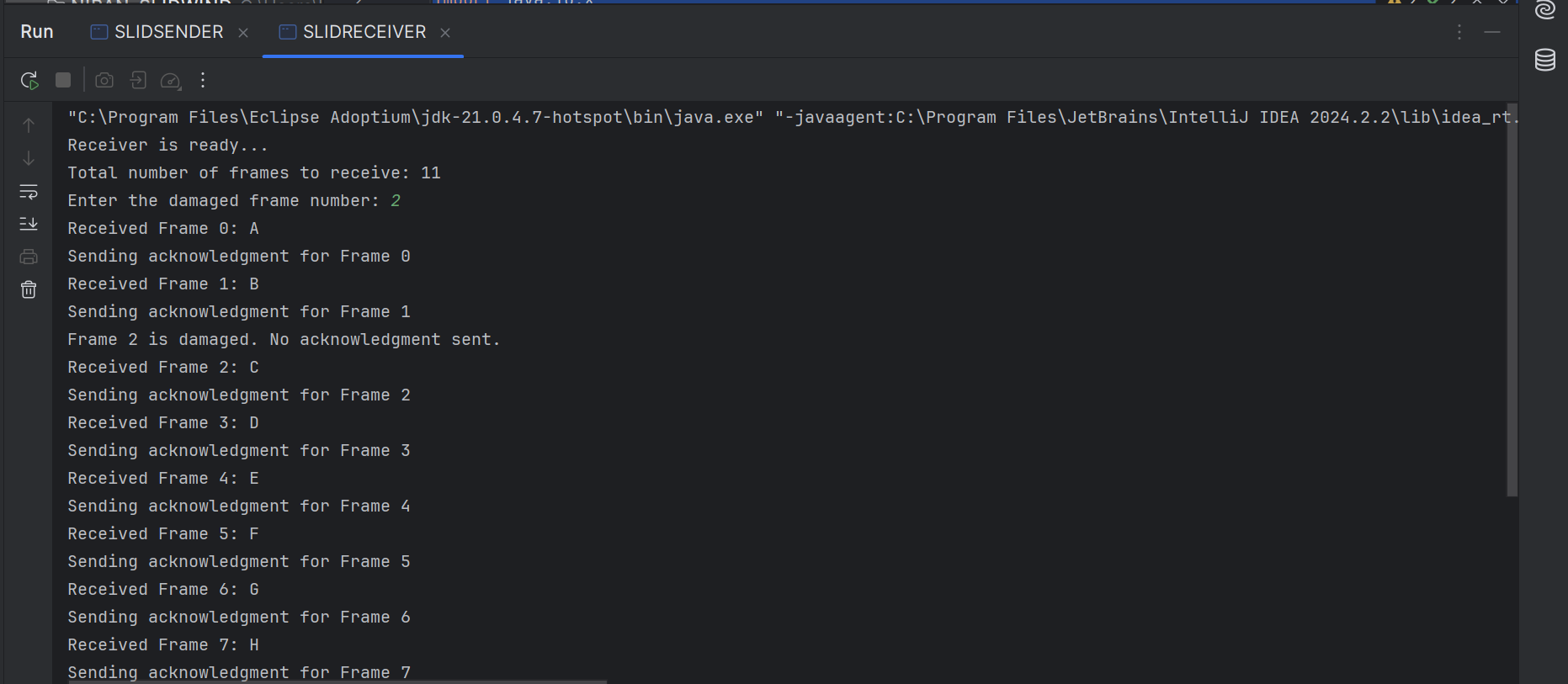
**Sender:**

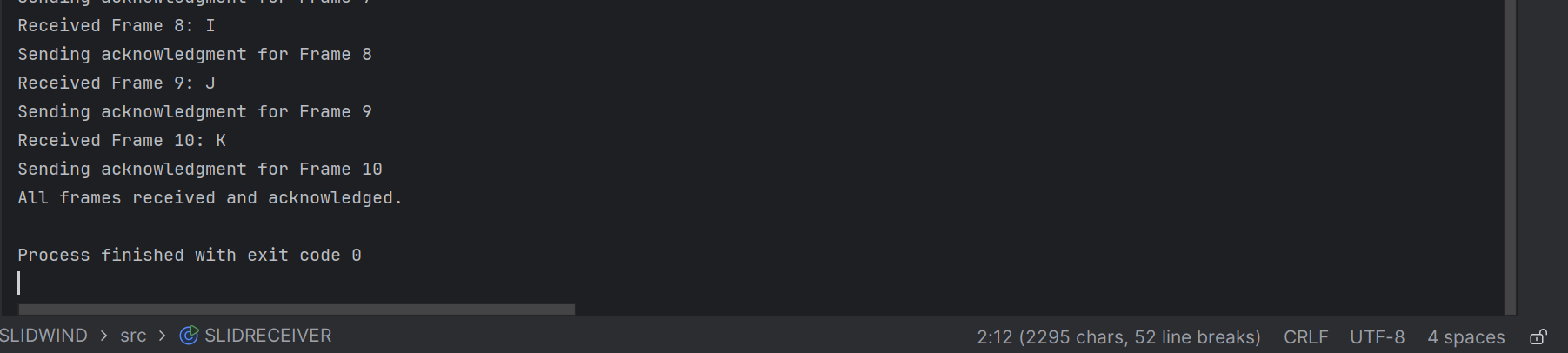
****

****

****

**Receiver:**

****

****

**Result:**

Thus Go-Back-N sliding window protocol has been simulated successfully in Java IDE.

* 1. **Selective Repeat window protocol**

**Theory**

• In sliding window method, multiple frames are sent by sender at a time before needing an acknowledgment.

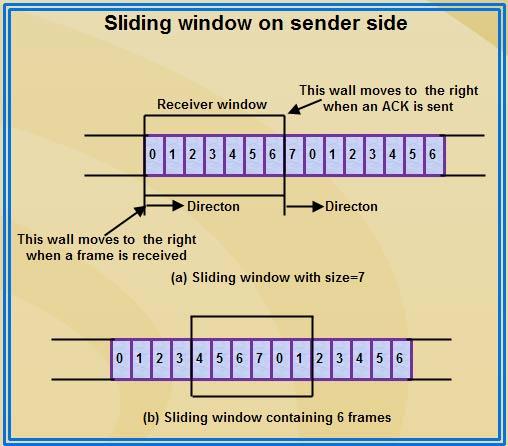
• Multiple frames sent by source are acknowledged by receiver using a single ACK frame.

Sender:

At the beginning of a transmission, the sender's window contains n-l frames.

• As the frames are sent by source, the left boundary of the window moves inward, shrinking the size of window. This means if window size is w, if four frames are sent by source after the last acknowledgment, then the number of frames left in window is w-4.

• When the receiver sends an ACK, the source's window expand i.e. (right boundary moves outward) to allow in a number of new frames equal to the number of frames acknowledged by that ACK.



Receiver:

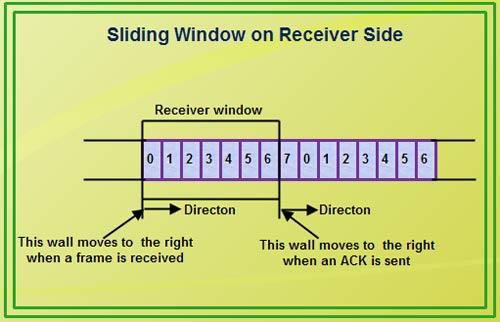
At the beginning of transmission, the receiver's window contains n-1 spaces for frame but not the frames.

• As the new frames come in, the size of window shrinks.

• Therefore the receiver window represents not the number of frames received but the number of frames that may still be received without an acknowledgment ACK must be sent.

• Given a window of size w, if three frames are received without an ACK being returned, the number of spaces in a window is w-3.

• As soon as acknowledgment is sent, window expands to include the number of frames equal to the number of frames acknowledged.

****

**Algorithm:**

**SENDER:**

**Initialization**:

* Create a ServerSocket and accept a connection.
* Initialize input/output streams for communication with the receiver.
* Create an array sbuff[] to store messages (frames).
* Create a boolean array ack[] to keep track of whether each frame has been acknowledged.
* Define variables for:
  + sws: the sliding window size
  + nf: the total number of frames
  + base: the lowest numbered frame in the window that has yet to be acknowledged
  + nextFrame: the next frame to be sent

**Get User Input**:

* Prompt the user to enter the window size (sws).
* Prompt the user to enter the total number of frames (nf).
* Send the total number of frames to the receiver.

**Read Frames**:

* For i from 0 to nf-1, do the following:
  + Read the message from the user and store it in sbuff[i].
  + Set ack[i] to false (indicating that the frame is not yet acknowledged).

S**ending Loop**:

* Repeat until all frames are acknowledged (base < nf):
  1. **Send Frames**:
     + While nextFrame < nf and the difference (nextFrame - base) < sws:
       - If ack[nextFrame] is false:
         * Print a message indicating the frame is being sent.
         * Send the frame (nextFrame + ":" + sbuff[nextFrame]) to the receiver.
       - Increment nextFrame.
  2. **Wait for Acknowledgment**:
     + Print a message indicating the sender is waiting for acknowledgment.
     + Record the current time (startTime).
     + Set timeout to 5000 milliseconds (5 seconds).
     + While the current time minus startTime is less than timeout:
       - If acknowledgment is ready to be read from the input stream:
         * Read the acknowledgment number (ano) from the receiver.
         * Print a message indicating which frame has been acknowledged.
         * Mark ack[ano] as true (the frame has been acknowledged).
         * **Slide the Window**:

While ack[base] is true (the base frame has been acknowledged):

Increment base (slide the window).

* + - * + If base is greater than or equal to nf, print a message indicating that all frames have been acknowledged and exit the loop.
  1. **Retransmit Unacknowledged Frames**:
     + For i from base to nextFrame - 1:
       - If ack[i] is false:
         * Print a message indicating the frame is being retransmitted.
         * Resend the unacknowledged frame (i + ":" + sbuff[i]) to the receiver.

**Close Connection**:

* After all frames are acknowledged, close the socket.

**RECEIVER:**

**Initialization**:

* Create a Socket to connect to the sender at a specified port.
* Initialize input/output streams for communication with the sender.
* Create an array rbuf[] to store received messages (frames).
* Create a boolean array received[] to track which frames have been received.
* Define variables for:
  + expectedFrame: the next frame number expected to be received.
  + totalFrames: the total number of frames to be received.
  + damagedFrame: simulate a damaged frame (e.g., frame number 2).
  + damagedFrameReceived: a flag to indicate whether the damaged frame has been received.

**Get Ready**:

* Print a message indicating that the receiver is ready to receive frames.

**Read Total Frames**:

* Read the total number of frames to be received from the sender.
* Print the total number of frames.

**Receiving Loop**:

* Repeat until all frames are received (expectedFrame < totalFrames):
  1. **Receive Frame**:
     + Read the incoming frame message from the sender.
     + Split the incoming message to extract the frame number and its content.
  2. **Check for Damaged Frame**:
     + If the received frame number equals damagedFrame and damagedFrameReceived is false:
       - Print a message indicating that the frame is damaged.
       - Set damagedFrameReceived to true.
       - **Skip Acknowledgment**: Continue to the next iteration of the loop without sending an acknowledgment.
  3. **Process Received Frame**:
     + If the frame number is greater than or equal to expectedFrame and it has not been received yet (received[frameNumber] is false):
       - Print a message indicating that the frame has been received.
       - Store the frame content in rbuf[frameNumber].
       - Mark the frame as received (received[frameNumber] = true).
       - **Send Acknowledgment**:
         * Print a message indicating that an acknowledgment is being sent for the received frame.
         * Send the frame number back to the sender as acknowledgment.
       - **Slide the Window**:
         * While received[expectedFrame] is true (the expected frame has been received):

Increment expectedFrame (slide the window to the next expected frame).

**Completion**:

* Once all frames are received, print a message indicating that all frames have been acknowledged.

**Close Connection**:

* Close the socket when done.

**Program:**

**SENDER:**

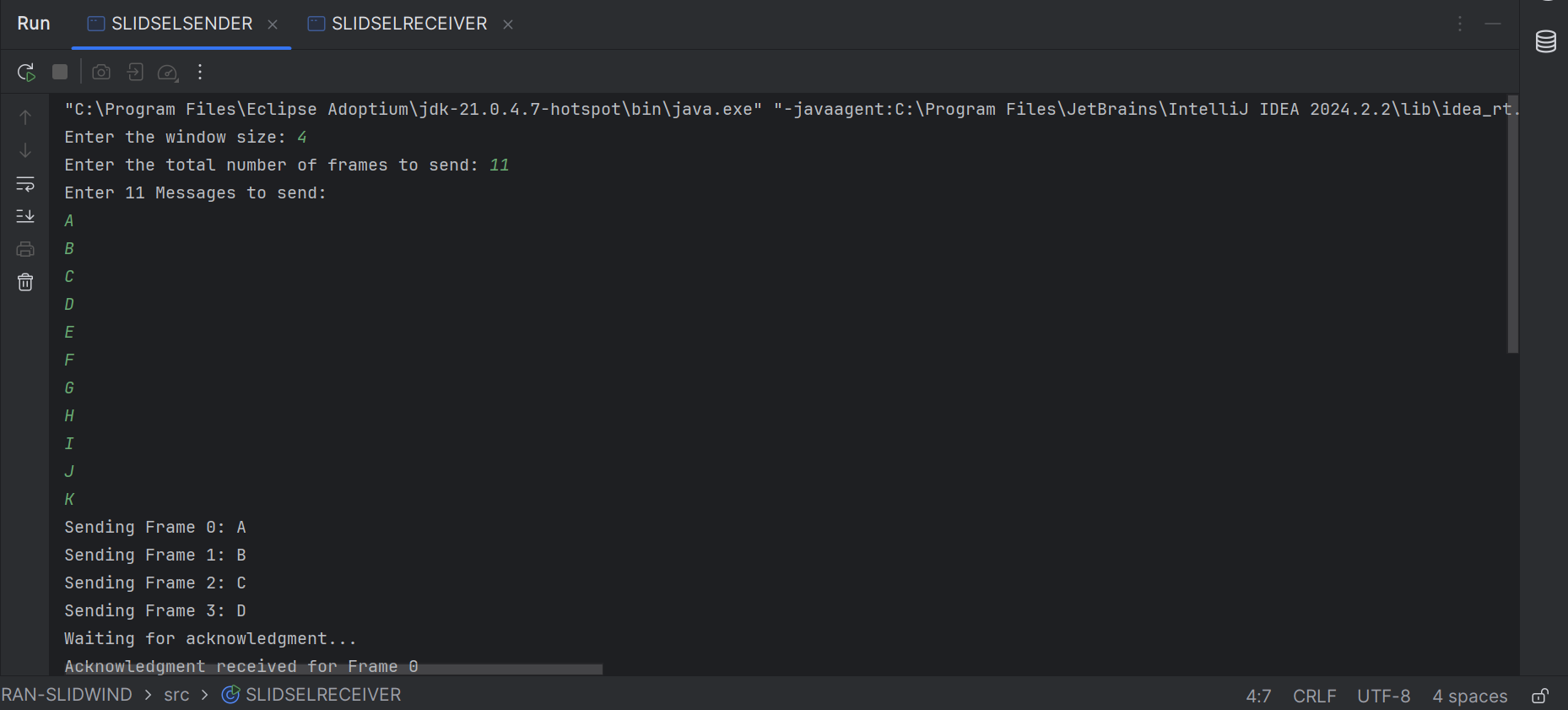
import java.net.\*;  
import java.io.\*;  
  
public class SLIDSELSENDER {  
 public static void main(String[] a) throws Exception {  
 ServerSocket ser = new ServerSocket(10);  
 Socket s = ser.accept();  
  
 BufferedReader in = new BufferedReader(new InputStreamReader(System.*in*));  
 BufferedReader in1 = new BufferedReader(new InputStreamReader(s.getInputStream()));  
 String sbuff[] = new String[100];  
 boolean ack[] = new boolean[100]; // To keep track of acknowledgments for each frame  
 PrintStream p;  
  
 int sws, nf, ano, i, base = 0, nextFrame = 0;  
  
 // Get the window size input from the user  
 System.*out*.print("Enter the window size: ");  
 sws = Integer.*parseInt*(in.readLine());  
  
 // Get the total number of frames  
 System.*out*.print("Enter the total number of frames to send: ");  
 nf = Integer.*parseInt*(in.readLine());  
  
 // Inform the receiver about the total number of frames  
 p = new PrintStream(s.getOutputStream());  
 p.println(nf);  
  
 // Reading all frames from the user and storing them in the buffer  
 System.*out*.println("Enter " + nf + " Messages to send:");  
 for (i = 0; i < nf; i++) {  
 sbuff[i] = in.readLine();  
 ack[i] = false; // Initially no frame is acknowledged  
 }  
  
 do {  
 // Send frames within the window  
 while (nextFrame < nf && (nextFrame - base) < sws) {  
 if (!ack[nextFrame]) { // Send only if the frame is not acknowledged  
 p = new PrintStream(s.getOutputStream());  
 System.*out*.println("Sending Frame " + nextFrame + ": " + sbuff[nextFrame]);  
 p.println(nextFrame + ":" + sbuff[nextFrame]); // Send frame number and message  
 }  
 nextFrame++;  
 }  
  
 // Waiting for acknowledgment  
 System.*out*.println("Waiting for acknowledgment...");  
  
 long startTime = System.*currentTimeMillis*();  
 int timeout = 5000; // 5 seconds timeout  
  
 while ((System.*currentTimeMillis*() - startTime) < timeout) {  
 // Check if acknowledgment is received  
 if (in1.ready()) {  
 ano = Integer.*parseInt*(in1.readLine());  
 System.*out*.println("Acknowledgment received for Frame " + ano);  
  
 // Mark the frame as acknowledged  
 ack[ano] = true;  
  
 // Slide the window base if base frame is acknowledged  
 while (ack[base]) {  
 base++;  
 }  
  
 // If all frames are acknowledged, exit the loop  
 if (base >= nf) {  
 System.*out*.println("All frames acknowledged.");  
 break;  
 }  
 }  
 }  
  
 // Check for unacknowledged frames and retransmit them  
 for (i = base; i < nextFrame; i++) {  
 if (!ack[i]) {  
 System.*out*.println("Retransmitting Frame " + i + ": " + sbuff[i]);  
 p.println(i + ":" + sbuff[i]); // Resend unacknowledged frame  
 }  
 }  
  
 } while (base < nf); // Continue until all frames are acknowledged  
  
 s.close();  
 }  
}

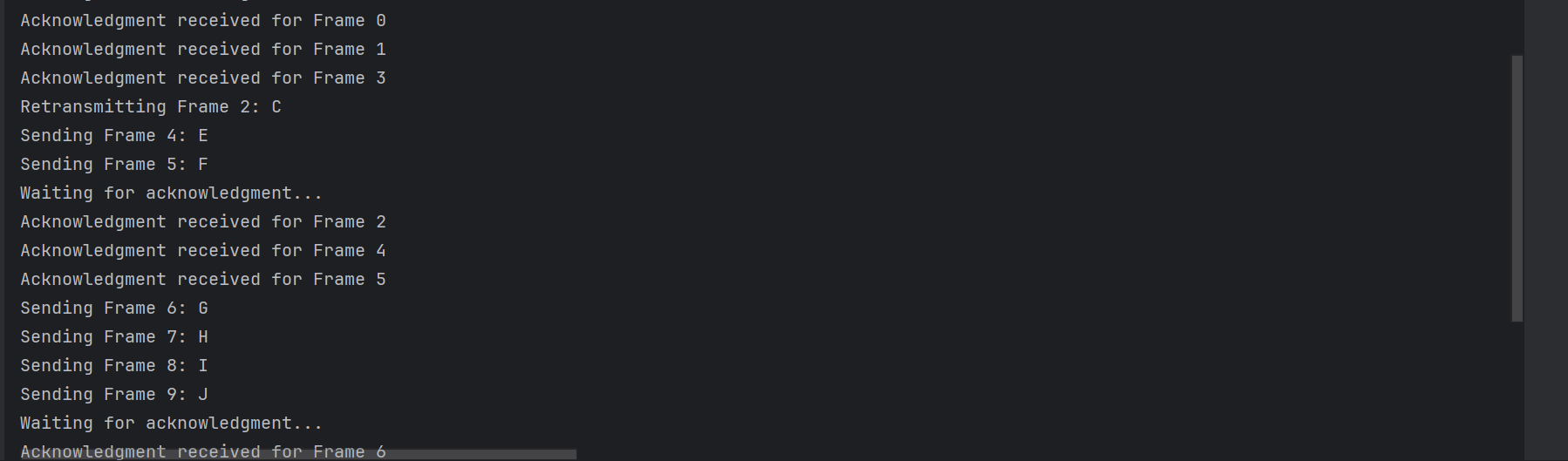
**RECEIVER:**

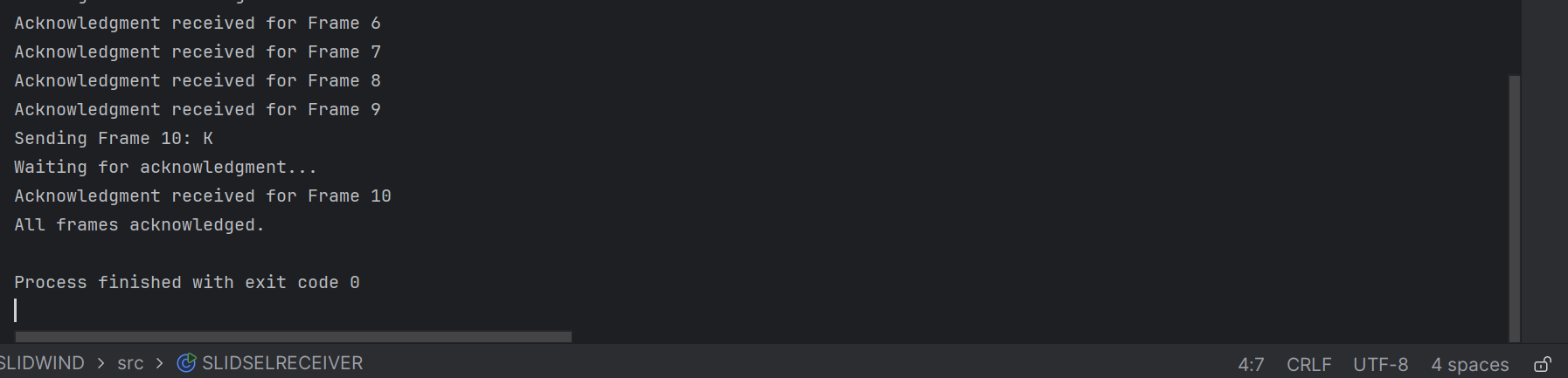
import java.net.\*;  
import java.io.\*;  
  
class SLIDSELRECEIVER {  
 public static void main(String[] a) throws Exception {  
 Socket s = new Socket(InetAddress.*getLocalHost*(), 10);  
 BufferedReader in = new BufferedReader(new InputStreamReader(s.getInputStream()));  
 PrintStream p = new PrintStream(s.getOutputStream());  
  
 int expectedFrame = 0, totalFrames;  
 String rbuf[] = new String[100];  
 boolean received[] = new boolean[100]; // To track which frames have been received  
 int damagedFrame = 2; // Assume frame 2 is damaged on first attempt  
 boolean damagedFrameReceived = false; // Flag to track if the damaged frame has been received  
  
 System.*out*.println("Receiver is ready...");  
  
 // Read the total number of frames to be received (sent by the sender)  
 totalFrames = Integer.*parseInt*(in.readLine());  
 System.*out*.println("Total number of frames to receive: " + totalFrames);  
  
 while (expectedFrame < totalFrames) {  
 // Receive the frame from the sender  
 String frameMessage = in.readLine();  
  
 // Split the incoming message to get the frame number and content  
 String[] parts = frameMessage.split(":");  
 int frameNumber = Integer.*parseInt*(parts[0]);  
 String frameContent = parts[1];  
  
 // Simulate frame damage for the first attempt  
 if (frameNumber == damagedFrame && !damagedFrameReceived) {  
 System.*out*.println("Frame " + frameNumber + " is damaged. No acknowledgment sent.");  
 damagedFrameReceived = true; // Mark that the damaged frame has been received  
 continue; // Don't send an acknowledgment for the damaged frame  
 }  
  
 // If the frame is in range and not already received  
 if (frameNumber >= expectedFrame && !received[frameNumber]) {  
 System.*out*.println("Received Frame " + frameNumber + ": " + frameContent);  
 rbuf[frameNumber] = frameContent; // Store the frame content  
 received[frameNumber] = true; // Mark the frame as received  
  
 // Send acknowledgment for the received frame  
 System.*out*.println("Sending acknowledgment for Frame " + frameNumber);  
 p.println(frameNumber);  
  
 // Slide the expected frame pointer to the next missing frame  
 while (received[expectedFrame]) {  
 expectedFrame++;  
 }  
 }  
 }  
  
 System.*out*.println("All frames received and acknowledged.");  
 s.close(); // Close the socket when done  
 }  
}

**Screenshot of output:**

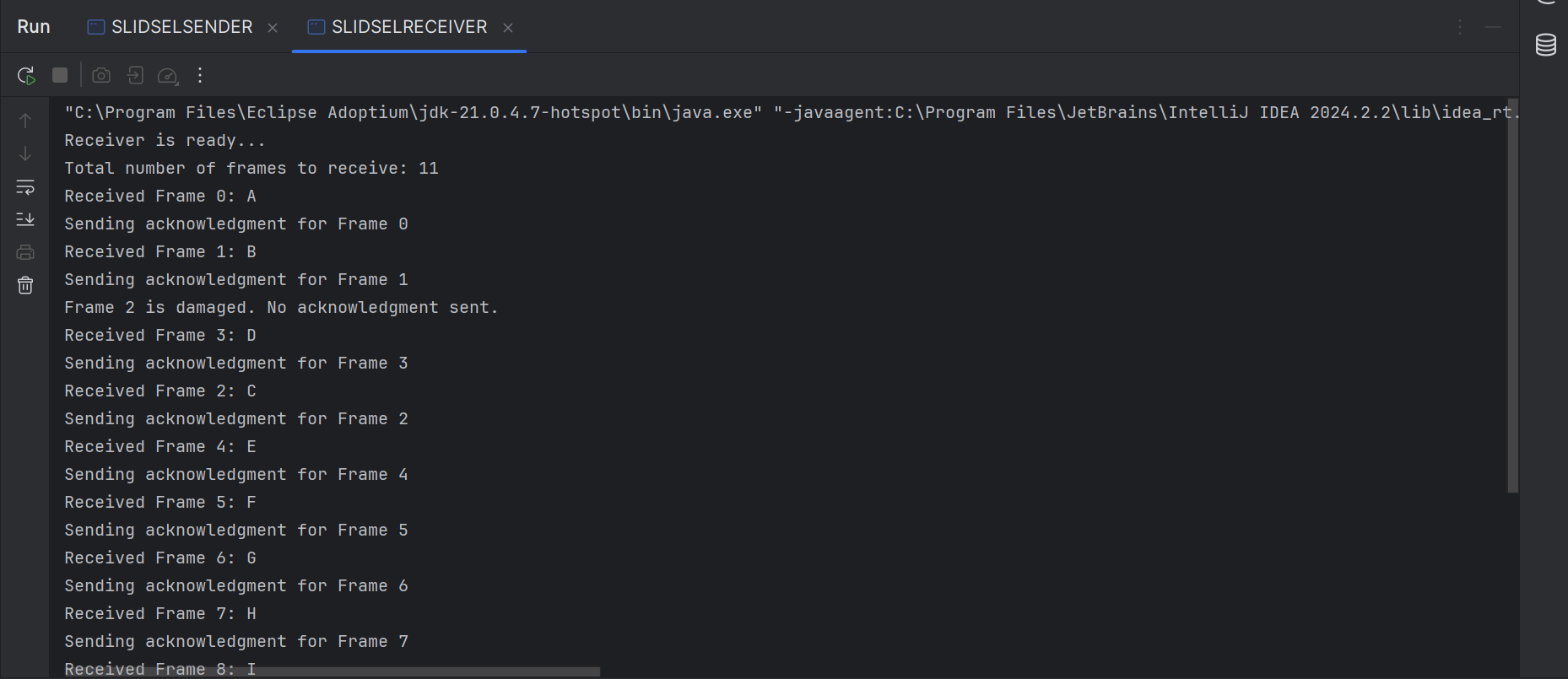
**Sender:**

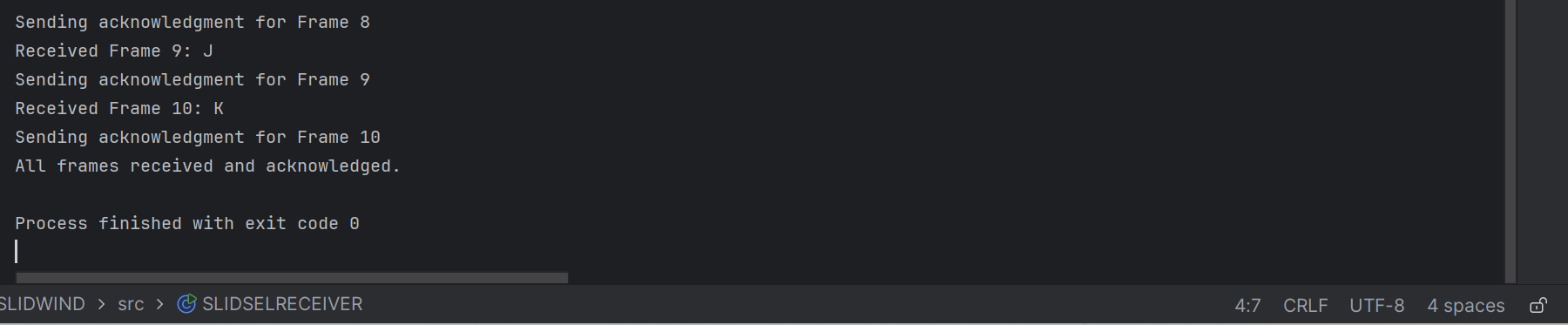
****

****

****

**Receiver:**

****

****

**Result:** Thus the Selective Repeat sliding window protocol is simulated successfully in Java IDE.