10. Develop a program for congestion control using leaky bucket algorithm.

```
import java.util.Scanner;  // Importing the Scanner class to take input from the user.
import java.lang.*;        // Importing the default Java language package (not strictly needed as it's
imported automatically).

public class leaky {
    public static void main(String[] args) {
        int i;                  // Declaring a variable 'i' for loop control.
        int a[] = new int[20];    // Declaring an array 'a' to store packet sizes, assuming a max of 20 packets.
        int buck_rem = 0;         // Remaining capacity in the bucket initially set to 0.
        int buck_cap = 4;         // Bucket capacity (max it can hold).
        int rate = 3;             // Transmission rate (rate at which packets are sent).
        int sent, recv;           // Variables to store sent and received packet sizes.

        Scanner in = new Scanner(System.in); // Scanner object to take user input.

        // Asking user to input the number of packets.
        System.out.println("Enter the number of packets");
        int n = in.nextInt();  // Reading the number of packets.

        // Asking the user to input the packet sizes.
        System.out.println("Enter the packets");
        for (i = 1; i <= n; i++)
            a[i] = in.nextInt();  // Reading the packet sizes into the array 'a'.

        // Printing the header for the output.
        System.out.println("Clock \t packet size \t accept \t sent \t remaining");

        // Simulating the packet transmission over time (clock cycles).
```

```
for (i = 1; i <= n; i++) {

    // Checking if the current packet is not zero.
    if (a[i] != 0) {
        // If the bucket doesn't have enough capacity to accept the incoming packet, mark it as dropped.
        if (buck_rem + a[i] > buck_cap)
            recv = -1;  // recv = -1 indicates packet is dropped.
        else {
            recv = a[i];  // Packet is accepted, store its size in recv.
            buck_rem += a[i];  // Update the remaining bucket capacity after accepting the packet.
        }
    }
    else {
        recv = 0;  // If packet size is 0, nothing is received.
    }

    // Now checking how much data can be sent from the bucket.
    if (buck_rem != 0) {
        // If the remaining data is less than the transmission rate, send all of it.
        if (buck_rem < rate) {
            sent = buck_rem;  // Send whatever is left.
            buck_rem = 0;     // After sending, the bucket becomes empty.
        } else {
            // Otherwise, send data at the fixed rate.
            sent = rate;
            buck_rem = buck_rem - rate;  // Reduce the remaining data by the rate.
        }
    }
    else {
        sent = 0;  // If bucket is empty, nothing is sent.
```

```
            }

            // If packet is dropped (recv == -1), print the dropped status.
            if (recv == -1)
                System.out.println(i + "\t\t" + a[i] + "\t dropped \t" + sent + "\t" + buck_rem);
            else
                // Otherwise, print the accepted, sent, and remaining values.
                System.out.println(i + "\t\t" + a[i] + "\t\t" + recv + "\t" + sent + "\t" + buck_rem);
        }
    }
}
```