```java
import java.io.BufferedReader;
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.FileReader;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.SocketTimeoutException;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.Random;

public class MySender {

	public static int TIMER = 3000;

	public static final double LOST_ACK_PROBABILITY = 0.05;

	public static final double BIT_ERROR_PROBABILITY = 0.1;

	public static void main(String[] args) {

		BufferedReader br = null;

		String fileName = "";
		int portNumber = 0;
		int numPackets = 0;

		String type = "";
		int sequenceNumBits = 0;
		int windowSize = 0;
		long timeOut = 0;
		long sizeSegment = 0;

		// getting the parameters

		if (args.length == 3) {
			fileName = args[0];
			portNumber = Integer.parseInt(args[1]);
			numPackets = Integer.parseInt(args[2]);

		} else {
			System.out.println("Invalid Parameters argv[0] - FileName (File containing configurations), argv[1] - PortNumber, argv[2] - NumberOfPackets\n");
		}

		try {
			br = new BufferedReader(new FileReader(fileName));
			String line = br.readLine();
			int i = 0;
			while (line != null) {
				if (i == 0) {
					type = line.trim();
				} else if (i == 1) {
					sequenceNumBits = Integer.parseInt(line.charAt(0) + "");
					windowSize = Integer.parseInt(line.charAt(2) + "");
				} else if (i == 2) {
					timeOut = Long.parseLong(line);

				} else if (i == 2) {
					timeOut = Long.parseLong(line);

				} else if (i == 3) {
					sizeSegment = Long.parseLong(line);

				}
				i++;
				line = br.readLine();
			}

			br.close();
		} catch (Exception e) {

			System.out.println("Error occured while reading file\n");
		}

		// Printing values obtained from file

		System.out.println("Type: " + type + " Number of Seq bits: " + sequenceNumBits + " Window Size " + windowSize
				+ " Timeout: " + timeOut + " Segment Size: " + sizeSegment + "\n");

		TIMER = (int) timeOut;

		// Sending Data Function
		try {
			sendData(portNumber, numPackets, type, sequenceNumBits, windowSize, timeOut, sizeSegment);
		} catch (Exception e) {

			e.printStackTrace();
		}

	}

	private static void sendData(int portNumber, int numPackets, String type, int sequenceNumBits, int windowSize,
			long timeOut, long sizeSegment) throws IOException, ClassNotFoundException, InterruptedException {

		ArrayList<SegmentData> sent = new ArrayList<>();

		// Last Packet sent
		int lastSent = 0;

		// Sequence number of the last Acknowledged packet
		int waitingForAck = 0;

		String alphabet = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ";
		int N = alphabet.length();

		DatagramSocket Socket = null;
		if (type.equalsIgnoreCase("gbn")) {

			byte[] incomingData = new byte[1024];
			InitiateTransfer initiateTransfer = new InitiateTransfer();
			initiateTransfer.setType(0);
			initiateTransfer.setNumPackets(numPackets);
			initiateTransfer.setPacketSize(sizeSegment);
			initiateTransfer.setWindowSize(1);

			Socket = new DatagramSocket();
			InetAddress IPAddress = InetAddress.getByName("localhost");

			ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
			ObjectOutputStream os = new ObjectOutputStream(outputStream);
			os.writeObject(initiateTransfer);
			byte[] data1 = outputStream.toByteArray();

			DatagramPacket initialPacket = new DatagramPacket(data1, data1.length, IPAddress, portNumber);
			System.out.println("Sending initial configuration to receiver" + "\n");
			Socket.send(initialPacket);

			DatagramPacket initialAck = new DatagramPacket(incomingData, incomingData.length);
			Socket.receive(initialAck);
			byte[] dataImp = initialAck.getData();
			ByteArrayInputStream inReturn = new ByteArrayInputStream(dataImp);
			ObjectInputStream isReturn = new ObjectInputStream(inReturn);
			InitiateTransfer initiateTransfer2 = (InitiateTransfer) isReturn.readObject();

			if (initiateTransfer2.getType() == 100) {

				while (true) {

					while (lastSent - waitingForAck < windowSize && lastSent < numPackets) {
						if (lastSent == 0 && waitingForAck == 0) {
							System.out.println("-Timer Started for Packet:  " + 0 + "\n");
						}
						Random r = new Random();
						char ch = alphabet.charAt(r.nextInt(N));
						int hashCode = ("" + ch).hashCode();
						SegmentData segmentData = new SegmentData();
						segmentData.setPayLoad(ch);
						segmentData.setSeqNum(lastSent);
						segmentData.setCheckSum(hashCode);
						if (lastSent == numPackets - 1) {
							segmentData.setLast(true);
						}

						if (Math.random() <= BIT_ERROR_PROBABILITY) {

							segmentData.setPayLoad(alphabet.charAt(r.nextInt(N)));

						}
						outputStream = new ByteArrayOutputStream();
						os = new ObjectOutputStream(outputStream);
						os.writeObject(segmentData);
						byte[] data = outputStream.toByteArray();

						DatagramPacket sendPacket = new DatagramPacket(data, data.length, IPAddress, portNumber);
						System.out.println("Sending Packet :" + segmentData.getSeqNum() + "\n");
						sent.add(segmentData);
						Socket.send(sendPacket);
						lastSent++;
						Thread.sleep(2500);

					}
					DatagramPacket incomingPacket = new DatagramPacket(incomingData, incomingData.length);
					try {
						Socket.setSoTimeout(TIMER);
						Socket.receive(incomingPacket);
						byte[] data = incomingPacket.getData();
						ByteArrayInputStream in = new ByteArrayInputStream(data);
						ObjectInputStream is = new ObjectInputStream(in);
						AckData ackData = (AckData) is.readObject();

						if (Math.random() > LOST_ACK_PROBABILITY) {
							System.out.println("Received ACK for :" + (ackData.getAckNo() - 1) + "\n");
							waitingForAck = Math.max(waitingForAck, ackData.getAckNo());
							if (!(waitingForAck == numPackets)) {
								System.out.println(
										"Timer Started for Packet:  " + ackData.getAckNo() + "\n");
							}
						} else {
							System.out.println("Acknowledgment Lost for :" + (ackData.getAckNo() - 1)
									+ "\n");
						}

						if (ackData.getAckNo() == numPackets) {
							break;
						}
					} catch (SocketTimeoutException e) {

						System.out.println(
								"Timeout Occured for Packet " + waitingForAck + "\n");

						for (int i = waitingForAck; i < lastSent; i++) {

							SegmentData segmentData = sent.get(i);
							char ch = segmentData.getPayLoad();
							int hashCode = ("" + ch).hashCode();
							segmentData.setCheckSum(hashCode);

							if (Math.random() <= BIT_ERROR_PROBABILITY) {
								Random r = new Random();
								segmentData.setPayLoad(alphabet.charAt(r.nextInt(N)));

							}
							outputStream = new ByteArrayOutputStream();
							os = new ObjectOutputStream(outputStream);
							os.writeObject(segmentData);
							byte[] data = outputStream.toByteArray();

							DatagramPacket sendPacket = new DatagramPacket(data, data.length, IPAddress, portNumber);
							System.out.println(
									"Re Sending Packet :" + segmentData.getSeqNum() + "\n");
							Socket.send(sendPacket);
							Thread.sleep(3000);

						}

					}
```

```
                        }
                }

        }

        else if (type.equalsIgnoreCase("sr")) {

                HashSet<Integer> unOrdered = new HashSet<>();
                byte[] incomingData = new byte[1024];
                InitiateTransfer initiateTransfer = new InitiateTransfer();
                initiateTransfer.setType(1);
                initiateTransfer.setNumPackets(numPackets);
                initiateTransfer.setPacketSize(sizeSegment);
                initiateTransfer.setWindowSize(windowSize);

                Socket = new DatagramSocket();
                InetAddress IPAddress = InetAddress.getByName("localhost");

                ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
                ObjectOutputStream os = new ObjectOutputStream(outputStream);
                os.writeObject(initiateTransfer);
                byte[] data1 = outputStream.toByteArray();

                DatagramPacket initialPacket = new DatagramPacket(data1, data1.length, IPAddress, portNumber);
                System.out.println("Sending Initial Data" + "\n");
                Socket.send(initialPacket);

                DatagramPacket initialAck = new DatagramPacket(incomingData, incomingData.length);
                Socket.receive(initialAck);
                byte[] dataImp = initialAck.getData();
                ByteArrayInputStream inReturn = new ByteArrayInputStream(dataImp);
                ObjectInputStream isReturn = new ObjectInputStream(inReturn);
                InitiateTransfer initiateTransfer2 = (InitiateTransfer) isReturn.readObject();

                if (initiateTransfer2.getType() == 100) {

                        while (true) {

                                while (lastSent - waitingForAck < windowSize && lastSent < numPackets) {

                                        /*if (lastSent == 0 && waitingForAck == 0) {
                                                System.out.println("!!!!! Timer Started for Packet: " + 0);
                                        }*/

                                        if(lastSent-waitingForAck==0)
                                        {
                                                System.out.println("Timer Started for Packet: " + lastSent + "\n");
                                        }
                                        else
                                        {
                                                System.out.println("Timer Already Running\n");
                                        }


                                        Random r = new Random();
                                        char ch = alphabet.charAt(r.nextInt(N));
                                        int hashCode = ("" + ch).hashCode();
                                        SegmentData segmentData = new SegmentData();
                                        segmentData.setPayLoad(ch);
                                        segmentData.setSeqNum(lastSent);
                                        segmentData.setCheckSum(hashCode);
                                        if (lastSent == numPackets - 1) {
                                                segmentData.setLast(true);
                                        }

                                        if (Math.random() <= BIT_ERROR_PROBABILITY) {

                                                segmentData.setPayLoad(alphabet.charAt(r.nextInt(N)));

                                        }
                                        outputStream = new ByteArrayOutputStream();
                                        os = new ObjectOutputStream(outputStream);
                                        os.writeObject(segmentData);
                                        byte[] data = outputStream.toByteArray();

                                        DatagramPacket sendPacket = new DatagramPacket(data, data.length, IPAddress, portNumber);
                                        System.out.println("Sending Packet : " + segmentData.getSeqNum() + "\n");
                                        sent.add(segmentData);
                                        Socket.send(sendPacket);
                                        lastSent++;
                                        Thread.sleep(2500);

                                }
                                DatagramPacket incomingPacket = new DatagramPacket(incomingData, incomingData.length);
                                try {
                                        Socket.setSoTimeout(TIMER);
                                        Socket.receive(incomingPacket);
                                        byte[] data = incomingPacket.getData();
                                        ByteArrayInputStream in = new ByteArrayInputStream(data);
                                        ObjectInputStream is = new ObjectInputStream(in);
                                        AckData ackData = (AckData) is.readObject();

                                        if (Math.random() > LOST_ACK_PROBABILITY) {
                                                System.out.println("Received ACK for :" + (ackData.getAckNo() - 1) + "\n");
                                                if ((ackData.getAckNo() - waitingForAck) == 1) {
                                                        waitingForAck = waitingForAck + 1;
                                                        if (unOrdered.size() > 0) {
                                                                for (int i = waitingForAck; i <= lastSent; i++) {

                                                                        if (unOrdered.contains(i)) {
                                                                                unOrdered.remove(i);
                                                                                waitingForAck++;
                                                                        } else {
                                                                                break;
                                                                        }

                                                                }
                                                        }
                                                        System.out.println("Timer Started for Packet:  " + waitingForAck + "\n");

                                                } else {
                                                        System.out.println("Timer already Running for   " + waitingForAck + "\n");
                                                        unOrdered.add((ackData.getAckNo() - 1));
                                                }


                                        } else {
                                                System.out.println("Acknowledgment Lost for :" + (ackData.getAckNo() - 1)
                                                                + "\n");
                                        }

                                        if (waitingForAck == numPackets && unOrdered.size() == 0) {
                                                break;
                                        }
                                } catch (SocketTimeoutException e) {

                                        System.out.println("Timeout Occured\n");

                                        for (int i = waitingForAck; i < lastSent; i++) {

                                                SegmentData segmentData = sent.get(i);
                                                if (!(unOrdered.contains(segmentData.getSeqNum()))) {

                                                        char ch = segmentData.getPayLoad();
                                                        int hashCode = ("" + ch).hashCode();
                                                        segmentData.setCheckSum(hashCode);

                                                        if (Math.random() <= BIT_ERROR_PROBABILITY) {
                                                                Random r = new Random();
                                                                segmentData.setPayLoad(alphabet.charAt(r.nextInt(N)));

                                                        }
                                                        outputStream = new ByteArrayOutputStream();
                                                        os = new ObjectOutputStream(outputStream);
                                                        os.writeObject(segmentData);
                                                        byte[] data = outputStream.toByteArray();

                                                        DatagramPacket sendPacket = new DatagramPacket(data, data.length, IPAddress,
                                                                        portNumber);
                                                        System.out.println(
                                                                        "Re Sending Packet :" + segmentData.getSeqNum() + "\n");
                                                        Socket.send(sendPacket);
                                                        Thread.sleep(2000);

                                                }

                                        }

                                }
                        }
                }

        }

}
```