**Final Project Report –Sockets Methods**

**Description of design:**

The producer process is responsible for creating a ServerSocket and specifying the port number as 8888. It then creates another socket to handle data transportation. The consumer only needs to create a single socket with the provided port information to receive the data. Last but not least, the ServerSocket and socket should be manually closed.

**System calls used for Producer:**

**(1)** Line22: writer.write(Integer.toString(num)):

Write() is an instance method from the BufferedWriter class. It's not a direct system call however underlying system calls may occur when writing to certain output destinations to a file.

**(2)** Line23: writer.newLine().Again, write() system call.

**(3)** Line25: writer.close():Close() system call.

**(4)** Line32: Socket socket = serverSocket.accept(): the accept() system call allows the server program to block and wait until a client connection is made.Once the connect is made, it will return a Socket object.

**(5)** Line39:printWriter.println(nums): writes the nums variable to the output stream using the println() method of the PrintWriter class.

**(6)** Line41: socket.close(): It closes the socket, it is a close system call.

**System calls used for Consumer:**

**(1)** Line13: Socket socket = new Socket("localhost",8888): It creates a client-side socket and attempts to connect to a server, which will involve underlying system calls like socket(),connect().

**(2)** Line16: BufferedReader reader = new BufferedReader(new InputStreamReader(socket.getInputStream())):Although it is not a direct system call, it will invoke underlying system calls for handling sockets.

**(3)** Line28:BufferedWriter writer = new BufferedWriter(new FileWriter("Consumer\_numbers.txt")); It is not a direct system call, it will invoke underlying system calls such as open() or create() in the constructor.

**(4)** Line33: writer.close():It closes the file, which corresponds to the close() system call.

**(5)** Line37: BufferedReader reader2 = new BufferedReader(new FileReader("Consumer\_numbers.txt")):It opens a file for reading, which would again involve system calls for file operations like open().

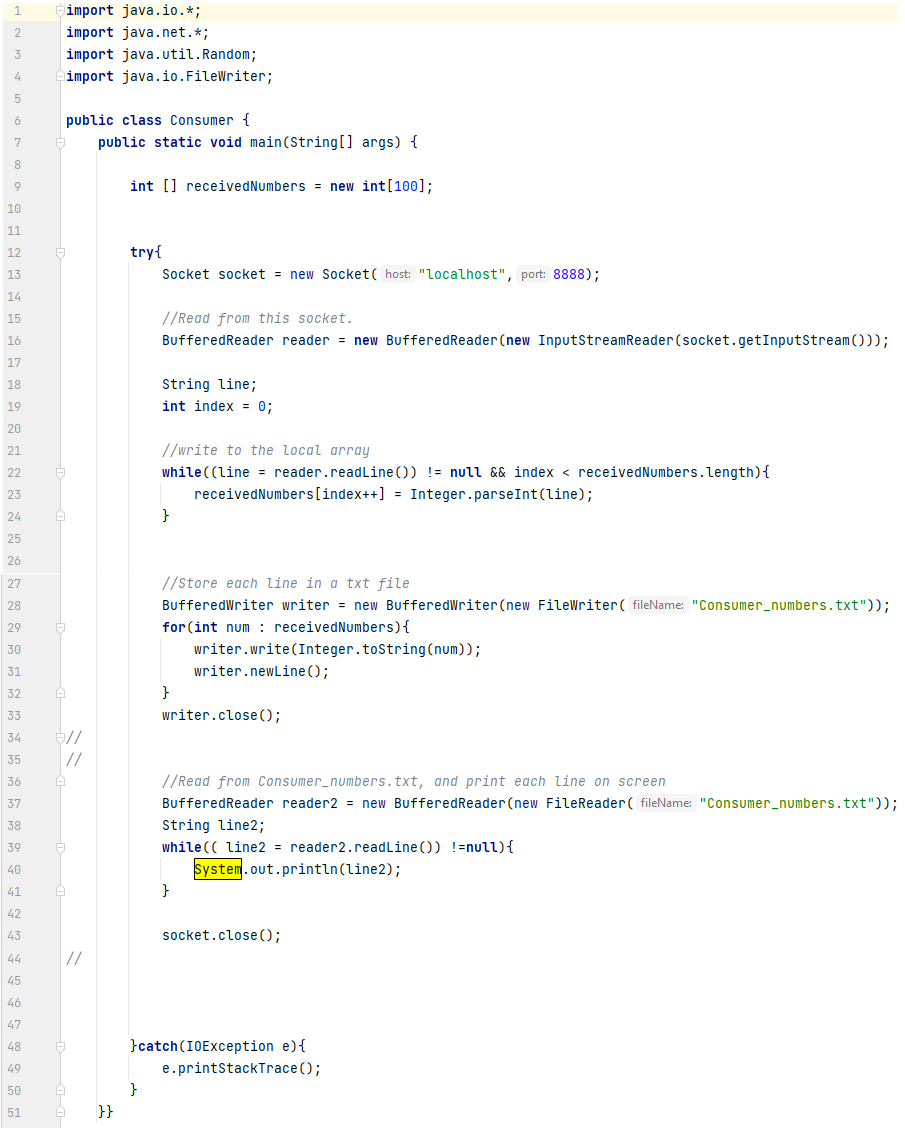
**(6)** Line39: line2 = reader2.readLine():It reads data from the file, which would involve the read() system call.

**(7)** Line40: System.out.println(line2):It will write data to the standard output, it will invoke the write() system call.

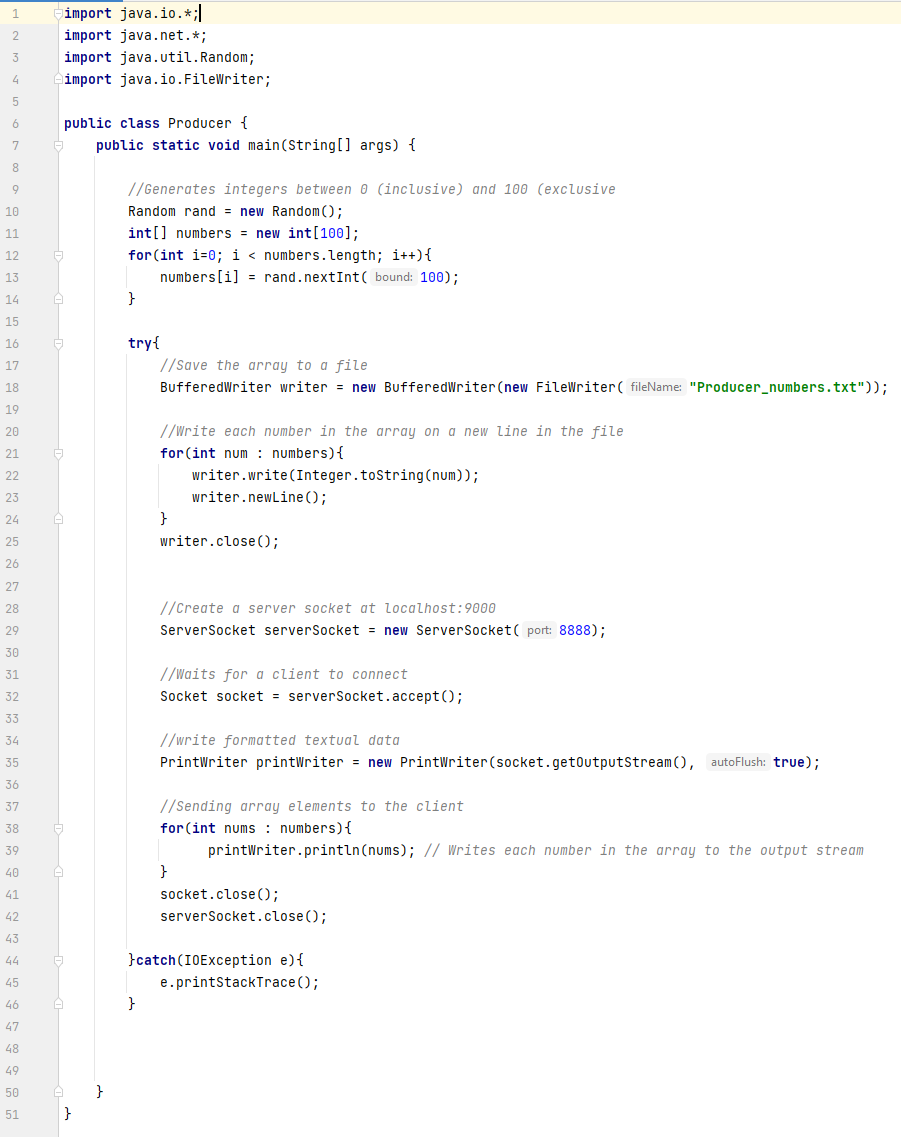
**(8)** Line43: socket.close():It closes the socket.

**Source code**:

**Consumer codes:**

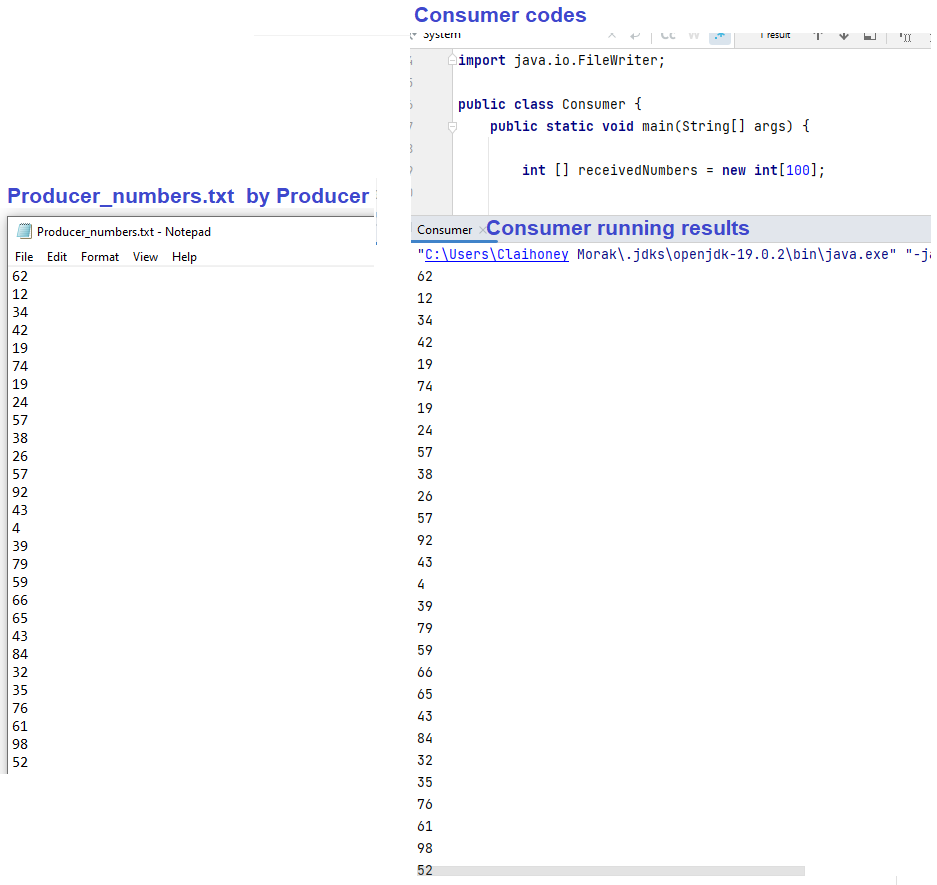


**Producer codes:**



**Result:**

So in I ran the Producer first in my IntelliJ IDE, then I open another IntelliJ IDE's window from which I ran the Consumer's process.For the Producer process it generate an integer array with 100 random number and is kept in Producer\_numbers.txt file. And the Producer pass the data of the array to the Consumer using socket IPC. After receiving the said array, all the data is kept in a Consumer\_numbers.txt file and then be printed out by the Consumer.



**Overall Discussion:**

Unlike the examples given in the handout materials, my project deals with processes running on the same computer. Therefore, instead of using an IP address as a parameter for the constructor, I use 'localhost' and a port number. Additionally, while the handout materials use an inner class called 'Connection' to handle a client connection, and automatically create two instances of 'DataInputStream' and 'DataOutputStream', my code differs. In my implementation, an 'OutputStream' is obtained from a socket instance passed as a parameter to the 'PrintWriter's' constructor. I chose to use 'PrintWriter' instead of 'DataOutputStream' because 'DataOutputStream' is a byte-oriented class, while 'PrintWriter' is a character-oriented class. Since I am dealing with strings, 'PrintWriter' is the better choice.