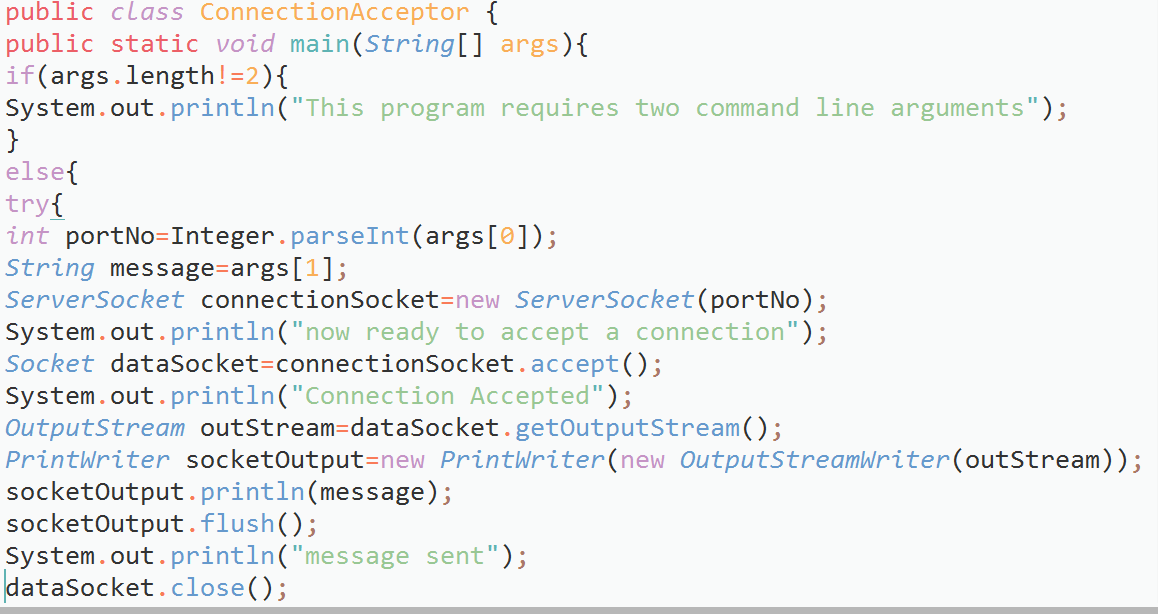
**Lab # 02**

**Objective:** To become familiar with Stream Socket API.

**Task 1:** Compile and run the above code. Start the acceptor first and then the requestor with appropriate command line arguments. Describe and explain the output.

Code:

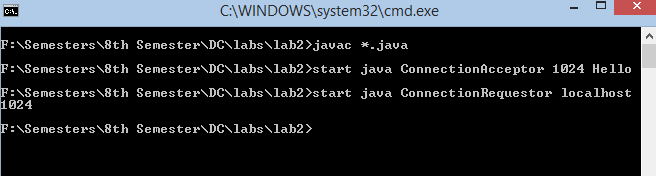
Acceptor:

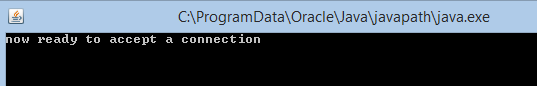
****

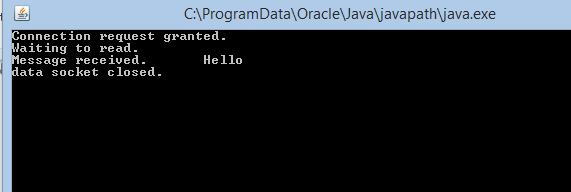
**Requestor:**

****

**Output:**

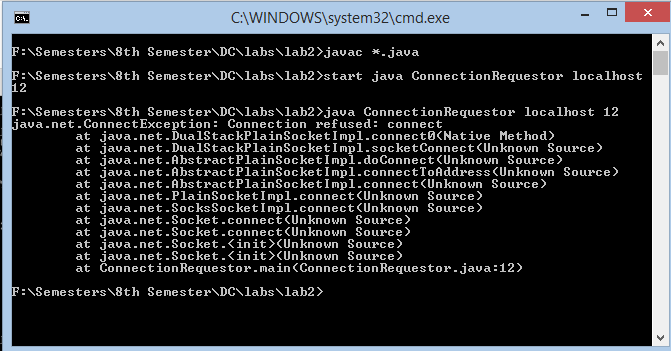
****

****

****

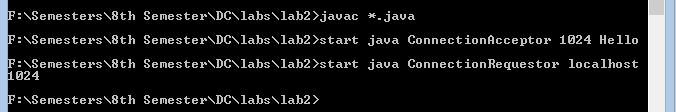
Description: First we start the acceptor that accepts the request and responds accordingly and then requestor that request any service. In above case acceptor starts and prepares its service that is a message in this case and then requestor is started in order to make a request that required the name and port number of acceptor in order to establish a connection and get the data or service after it receives the message the acceptor flushes all the data (some like session ends) and close the socket and ends the connection. We can say here the acceptor class is working as a server while requestor is working as receiver.

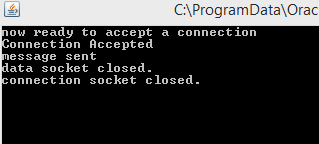
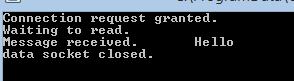
**Task2:** Now run the code again, but reverse the order of program’s execution. Start the requestor first and then the acceptor. Describe and explain the outcome.

****

Description: Exception occurs as there is no acceptor to accept the requestors connection.

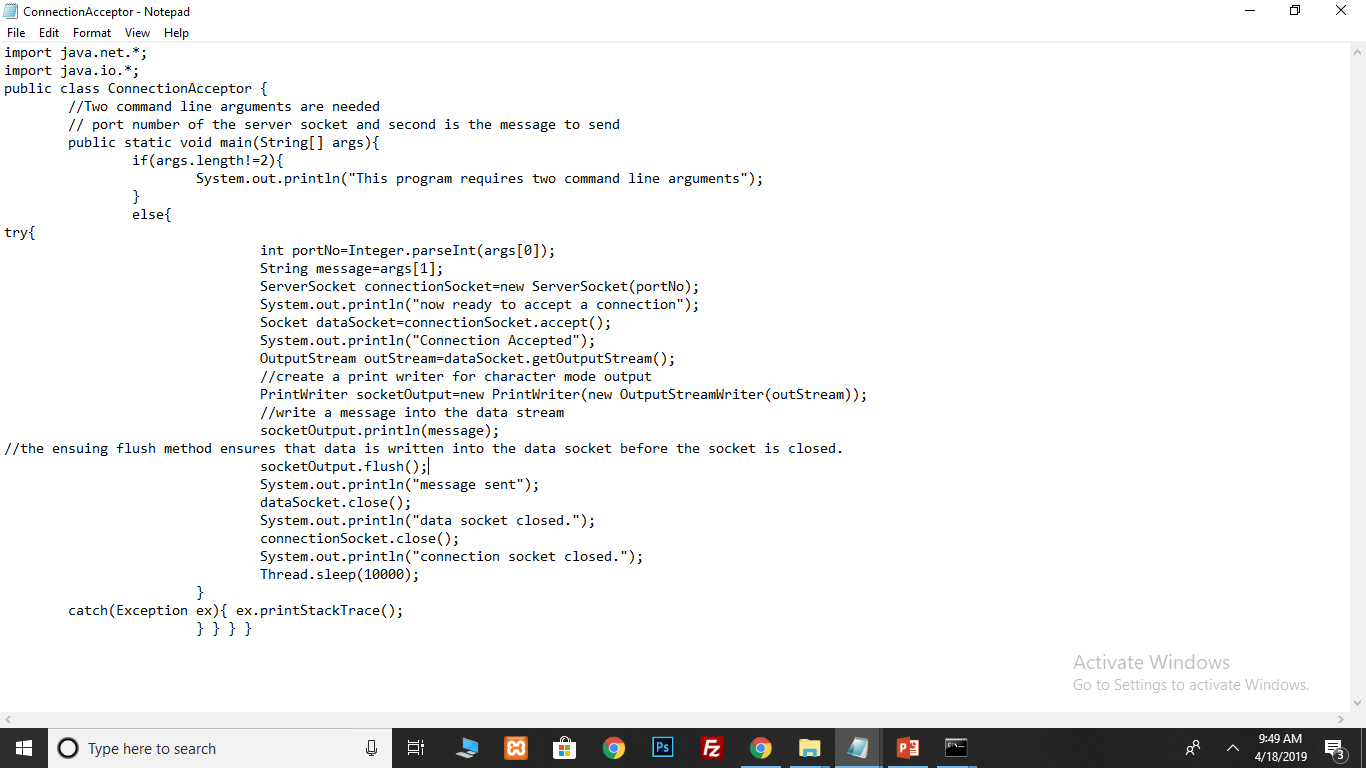
**Task 3**: Add a time delay of 5 seconds in the ConnectionAcceptor process just before the message is written to the socket, then run the program. This will show you the blocking at the receiver. Show a trace of the output of the processes.

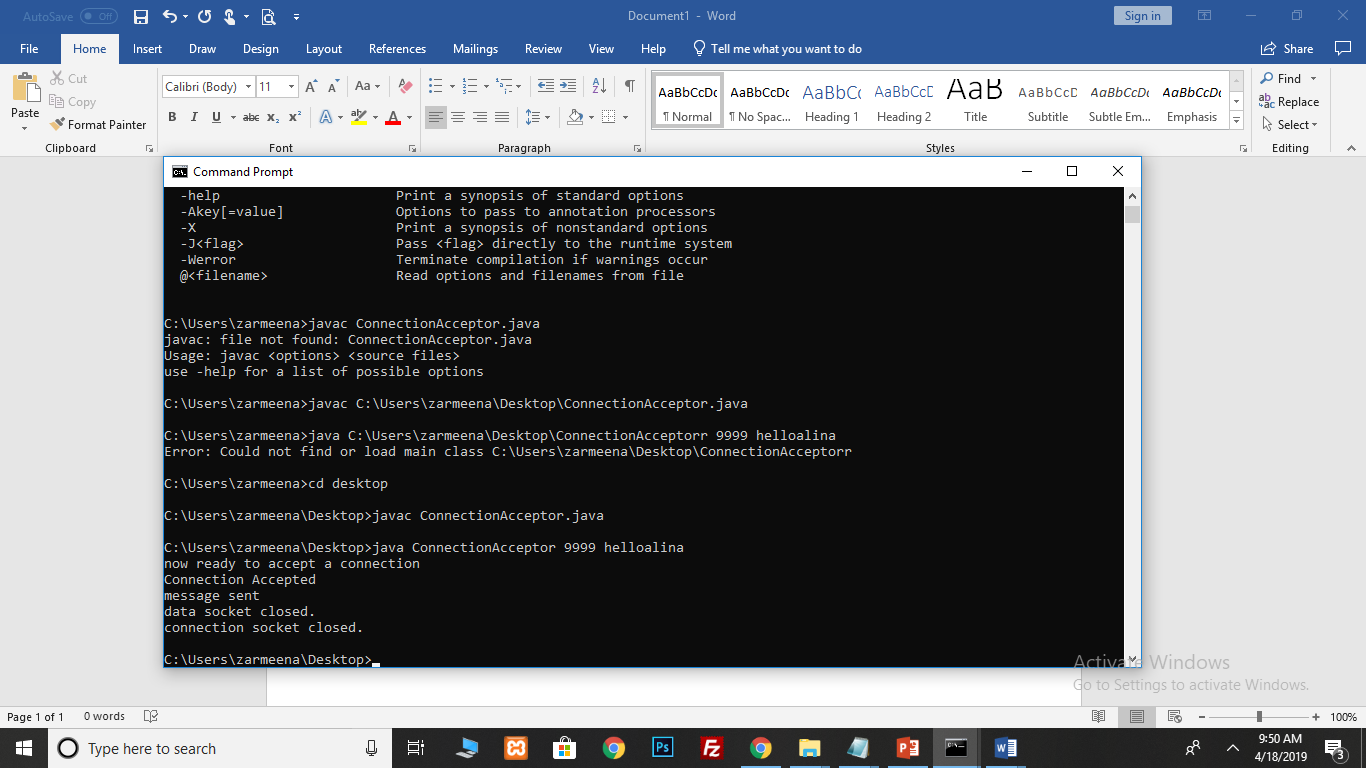
****

** **

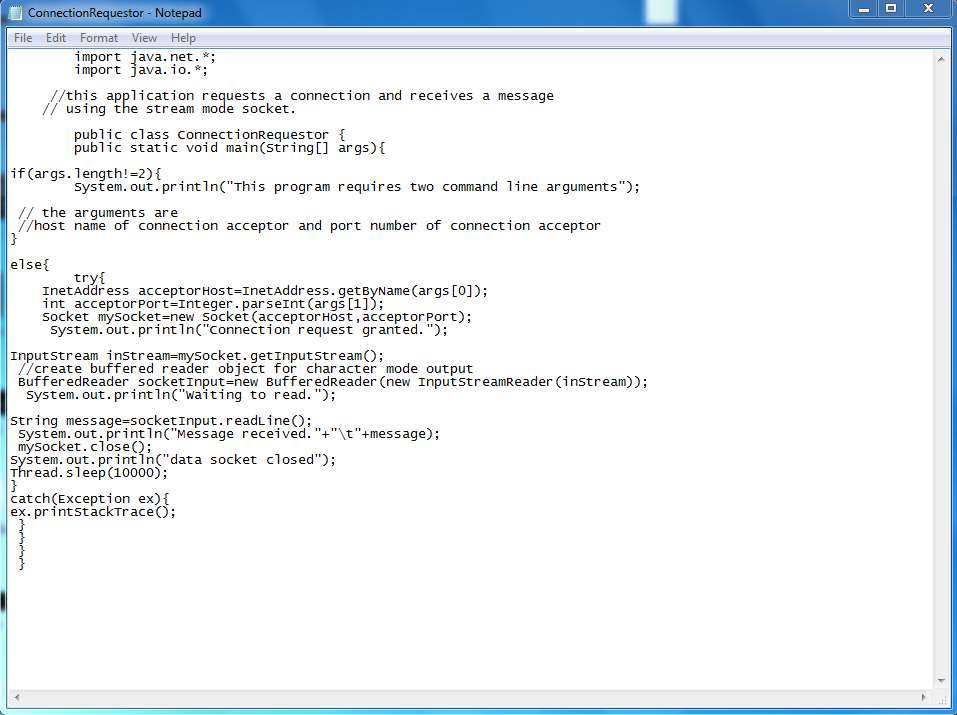
**Task4:** Modify the sample code to include two way communication between the client and the server.

Server

**Output**



**Client**



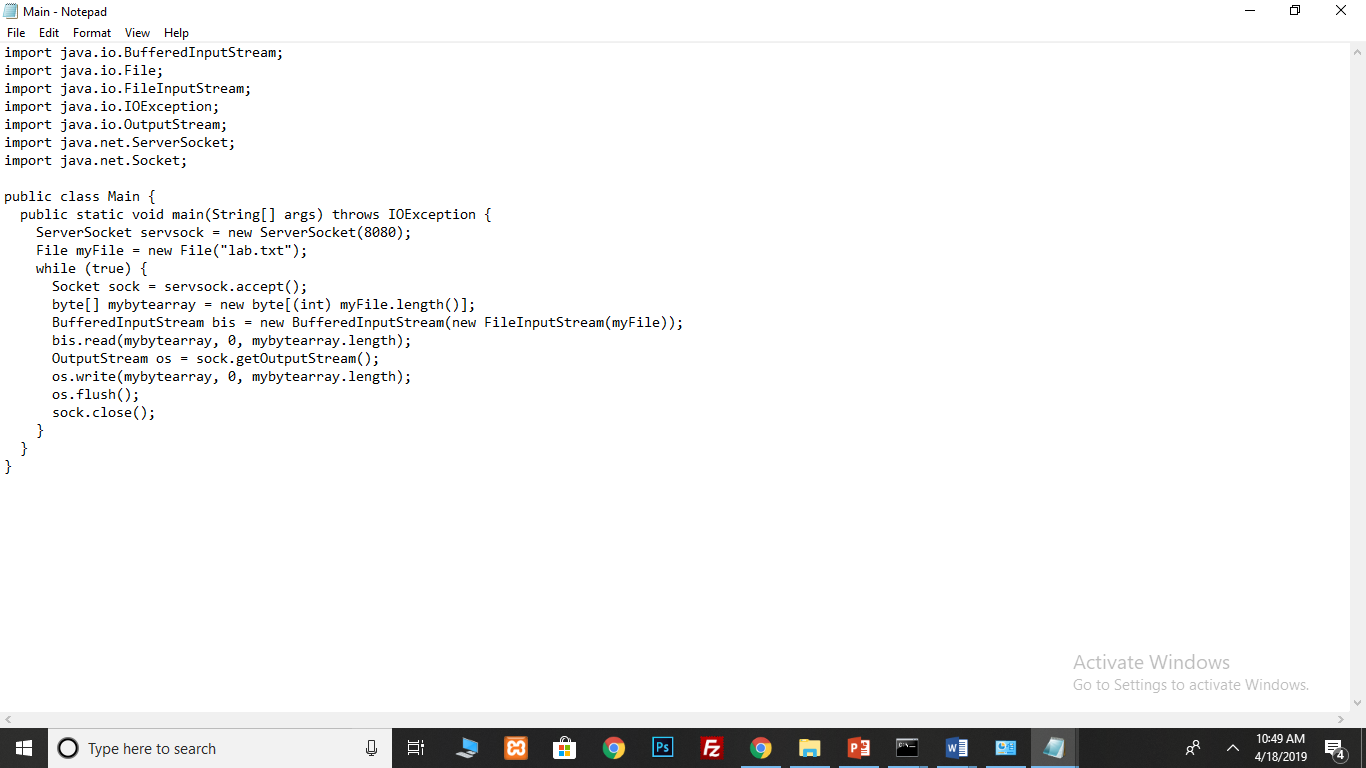
**Output**



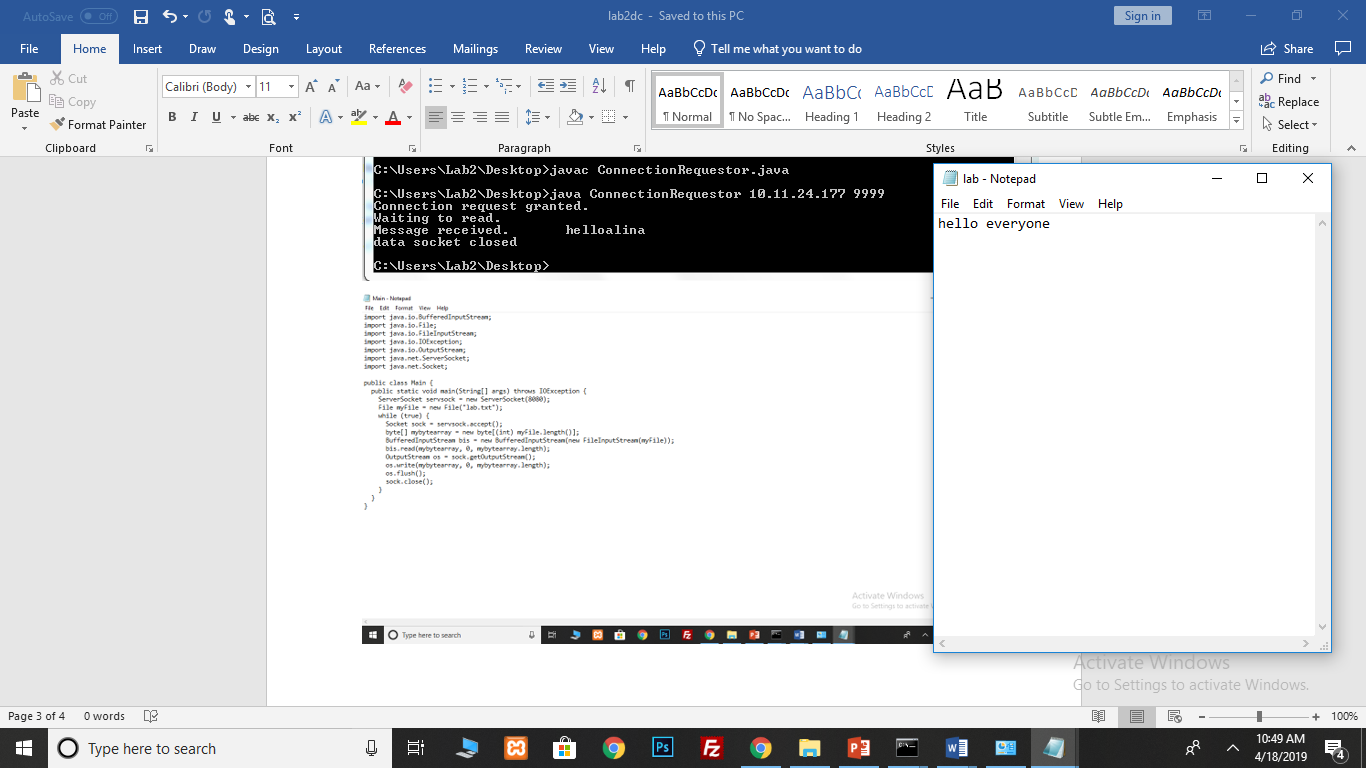
**Task 5:** Modify the sample code to send complete files between the client to the server.

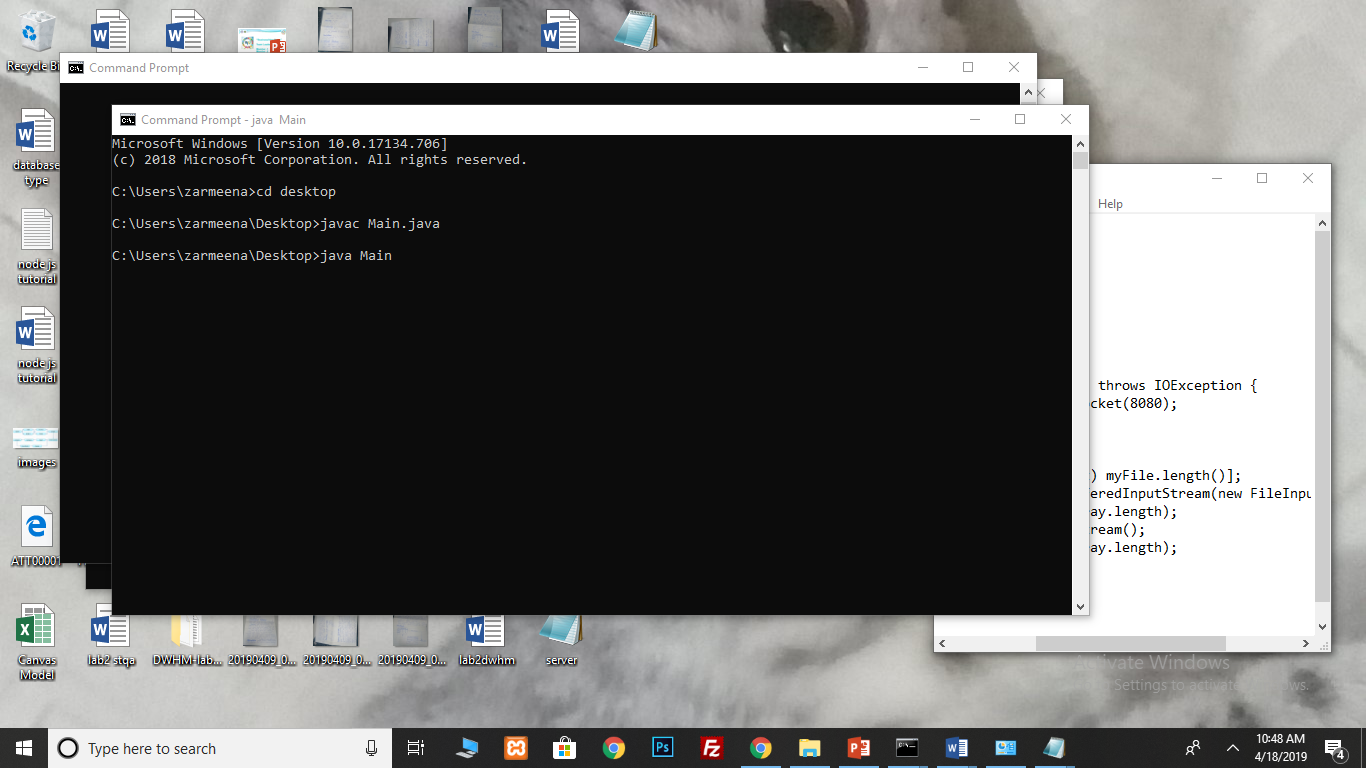
Explore the non-blocking

Server (sending file to client)

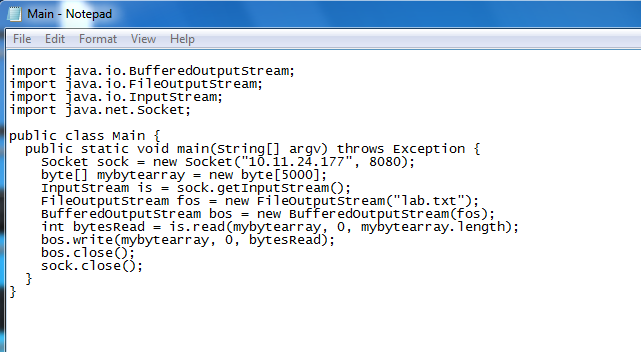


File in server coputer

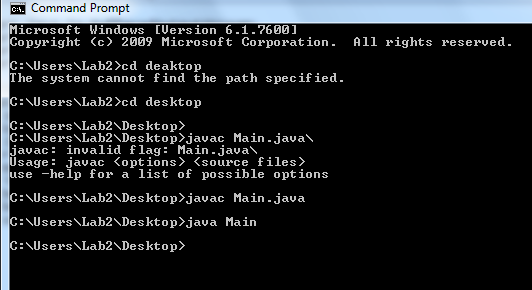


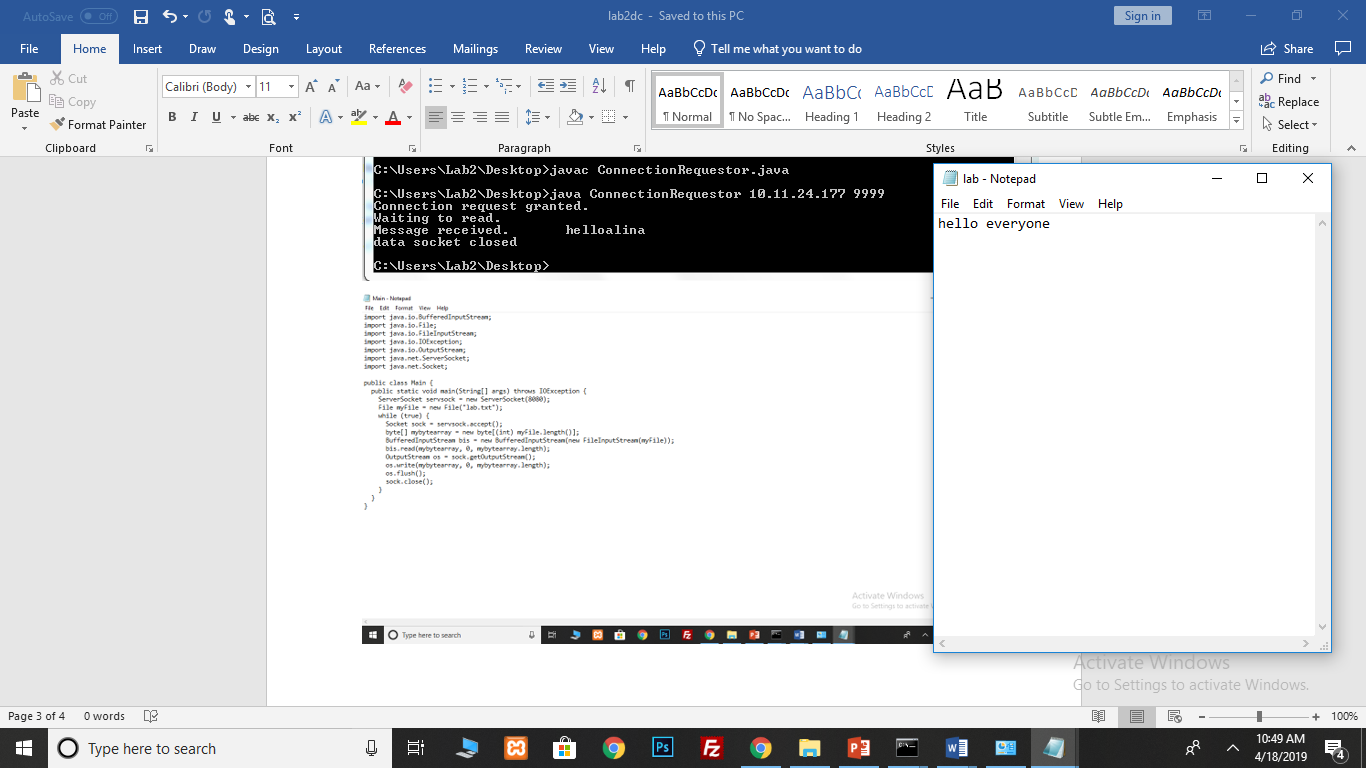


**Client**



**Output**



File received on client computer

**Task 6:** Explore the non-blocking java socket API in the **nio** package and implement a sample program.

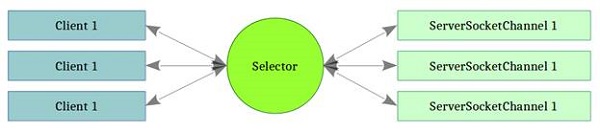
-Socket programs in java can be made to work in both blocking and non-blocking mode. In blocking socket mode, a system call event halts the execution until an appropriate reply has been received. In non-blocking sockets, it continues to execute even if the system call has been invoked and deals with its reply appropriately later.

|  |  |
| --- | --- |
| *Non-blocking socket classes* | *Description (\*)* |
| *ServerSocketChannel* | A selectable channel for stream-oriented listening sockets. The channel is created by invoking the *open* method of this class. This class uses the *ServerSocket* class behind the scenes. An instance of this class is used to accept a new connection request in a server like the *ServerSocket* class instance. |
| *SocketChannel* | A selectable channel for stream-oriented listening sockets. The channel is created by invoking the *open* method of this class. This class uses the *Socket* class behind the scenes. An instance of this class is used to establish connection between the client and the server like the *Socket* class instance. |
| *Selector* | A multiplexor for the *SelectableChannel* object. |
| *SelectionKey* | Used for representing the registration of *SelectableChannel* with a *Selector*. |

(\*) Refer to the Java API documentation for more information

**How It Works**

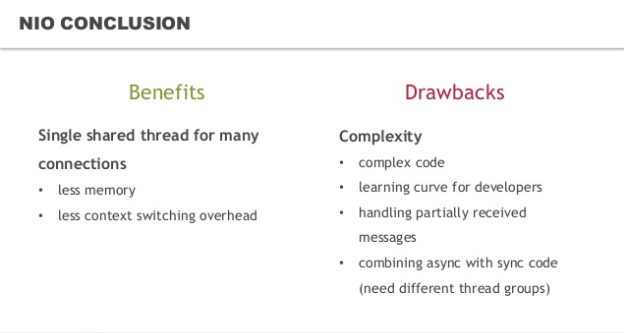
The Selector class object acts as an interface between remote clients and the server in the following manner.

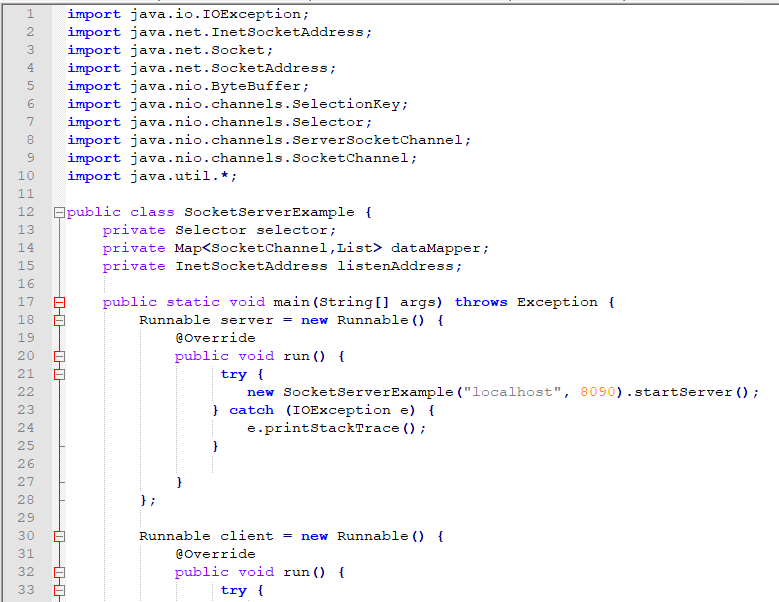


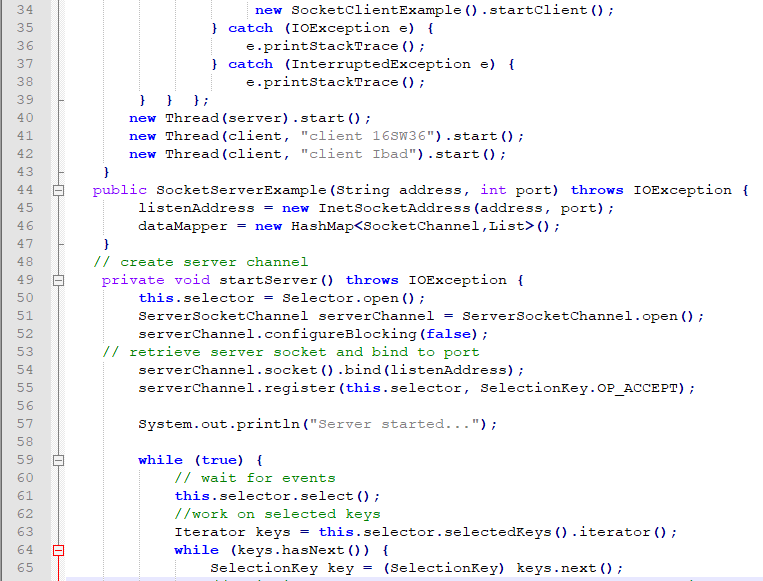
**Task6:** Explore the non-blocking java socket API in the **nio** package and implement a sample program.

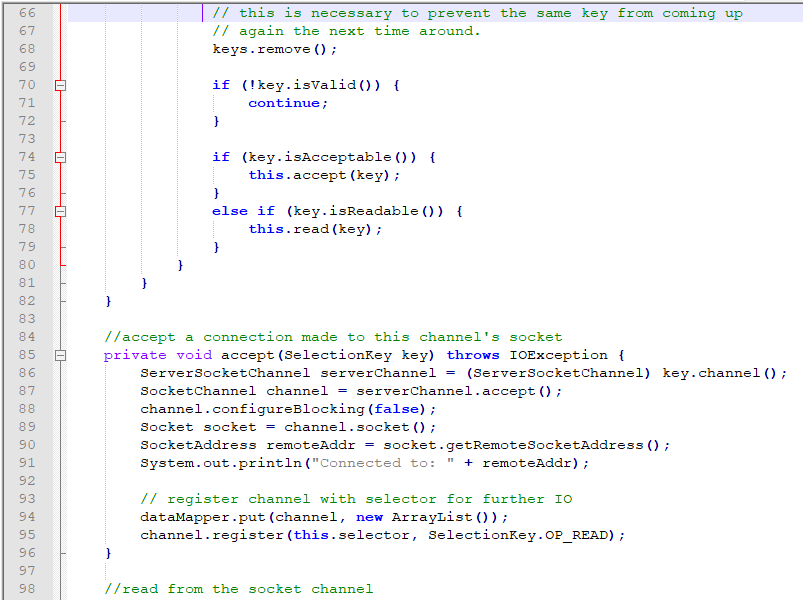
(nio advantages and disadvantages)

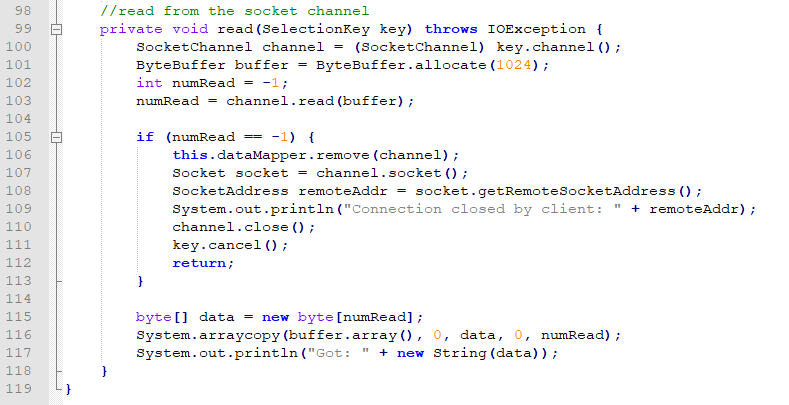
**Advantages Disadvantages**



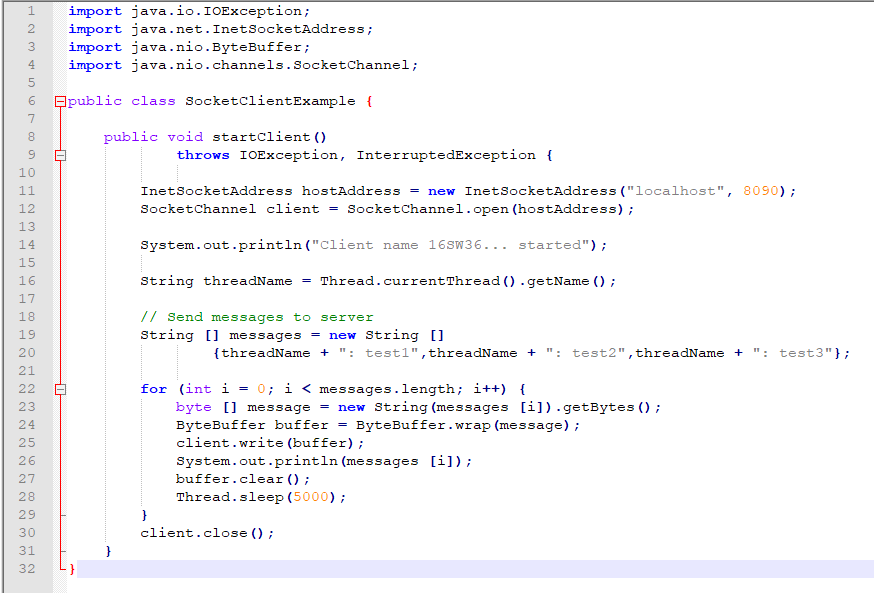
**Server** 







**Client**



**OUTPUT:**

