**Advanced Operating System**

**Project-2**

Submission by Kundanapreethi Surepally (2249580)

Table of Contents

[How to Execute: 3](#_Toc138746298)

[Classes: 6](#_Toc138746299)

[MainClass 6](#_Toc138746300)

[MiddleServer 7](#_Toc138746301)

[MiddleServerThread 8](#_Toc138746302)

[Sender 9](#_Toc138746303)

[MatchAPattern 10](#_Toc138746304)

[Receiver 11](#_Toc138746305)

# How to Execute:

To execute just run the Driver.class file inside bin folder and choose 3 in 1st menu.

> java com.pubsub.Driver 9098

Expected output:

A screenshot of a computer

Description automatically generated

This will start broker server in 9098 port and will also show the ip address it’s hosted in

To start publisher run the same program in another system with ip address of the broker as a runtime argument and choose 1 in 1st menu

> java Driver com.pubsub.Driver 192.168.1.18:9098

A screenshot of a computer

Description automatically generated

Now It will prompt to input username/password, on successful authentication, publisher will have access to tuple write and read rmi methods of broker

A screenshot of a computer

Description automatically generated

To add a new tuple and a song into it follow the below instructions.

A screenshot of a computer program

Description automatically generated

> java Driver com.pubsub.Driver 192.168.1.18:9098

Now start the subscriber program with similar command line argument as publisher

A screenshot of a computer program

Description automatically generated

To check the wallet balance:

A screenshot of a computer

Description automatically generated

# Classes:

## Driver

This is a driver class that represents a simple implementation of a publisher-subscriber pattern using a broker. It consists of a Driver class with a main method.

Here's a breakdown of the code:

1. The Driver class is defined within the com.pubsub package.
2. The necessary imports are included at the beginning of the code.
3. The main method is the entry point of the program.
4. It prompts the user to select a mode: Publisher, Subscriber, or Broker.
5. The user's choice is read using a Scanner object.
6. The chosen mode is processed using a switch statement.

* If mode 1 is selected, a Seller object is created and its startExecution() method is called. The Seller class is not shown in the provided code snippet.
* If mode 2 is selected, a Customer object is created and its startExecution() method is called. The Customer class is not shown in the provided code snippet.
* If mode 3 is selected, a Broker object is created, passing a Scanner object and an integer argument (Integer.parseInt(args[0])) to its constructor. Then, the startExecution() method of the Broker object is called.

1. If an exception occurs while creating the Broker object (RemoteException), the stack trace is printed.
2. If an invalid mode is selected, a corresponding message is displayed.
3. The Scanner object is closed.

## Broker

## MiddleServerThread

The MiddleServerThread class extends the SocketThread class and implements the Runnable interface. It represents a thread that handles client connections in the middle server mode of the network communication system.

Here's an overview of the class:

middleServer: A reference to the MiddleServer object that creates this thread.

receiverStr: A string variable to store the list of sender connections.

The class has the following methods:

MiddleServerThread(Socket s, DataInputStream dis, DataOutputStream dos, MiddleServer middleServer): The constructor that takes a Socket object, DataInputStream, DataOutputStream, and a MiddleServer object. It calls the constructor of the superclass (SocketThread) and assigns the middleServer parameter to the corresponding attribute.

run(): The run method is the entry point of the thread. It handles the client connection logic. Inside the method:

The method prompts the client to enter their credentials (username/password) and checks if the middle server has a matching user.

If the login is successful, a welcome message is sent to the client.

If the user has the role "sender", the thread expects the sender to send their connection details, which are then added to the middleServer.senderList. A response is sent back to the sender indicating the success or failure of the registration.

If the user has the role "receiver", the thread expects the receiver to send a request for the list of senders. The thread responds by sending the receiverStr, which contains the list of sender connections.

## Sender

The Sender class extends the common.Parent class and represents a sender in the network communication system. It has the following attribute:

registry: An instance of the Registry class from the RMI (Remote Method Invocation) framework.

The class has the following methods:

Sender(String startingPort): The constructor that takes a starting port as a parameter. It calls the constructor of the superclass (Parent) and initializes the registry attribute.

startExecution(): A public method that represents the main execution logic of the sender. It prints a message indicating that it is running in sender mode, and then calls the registerRMI() and connectToSocket() methods.

registerRMI(): A private method that registers the sender as an RMI object in the RMI registry. It creates a registry on the provided starting port, exports a PatternFinderRemote object as an RMI remote object, and binds it to the registry with a specified name (MyConst.REGISTRY\_NAME).

connectToSocket(): A private method that establishes a connection with the middle server. It prompts the user to enter the IP address and port of the middle server, creates a socket connection with the middle server, and sets up input and output streams for communication.

The method handles the login process by reading login prompts from the middle server and sending the user's credentials.

After successful login, the method sends the connection details of the sender (IP address and starting port) to the middle server for self-registration.

The method then enters a loop where it prompts the user for input. If the user enters "Exit", the connection is closed, and the loop is terminated.

Finally, the method closes resources, including the socket connection, input and output streams, and unbinds the RMI object from the registry.

## 

## MatchAPattern

This MatchAPattern class provides a method called perform that performs pattern matching on a predefined text file. Here's a breakdown of what the class does:

The perform method takes a pattern (p\_pattern) as input and returns a string containing the result of the pattern matching.

* It initializes a LinkedList called res to store the positions of the pattern matches.
* It creates a BufferedReader to read the predefined text file (specified by MyConst.MOON\_FILE).
* It reads the file line by line and performs pattern matching on each line.
* For each line, it searches for the pattern (case-insensitive) using the indexOf method.
* If a match is found, it creates a LinkedList called item to store the position of the match (index + 1) and the line number (lineNumber).
* It adds the item to the res list.
* It continues searching for the pattern in the line until no more matches are found.
* After processing all the lines in the file, it returns the result of the pattern matching by calling the printPatternResult method with the res list as the parameter.
* The printPatternResult method takes a list of lists (resultFound) containing the positions and line numbers of the pattern matches and generates a formatted string representing the result. It iterates over the resultFound list, formats each match's position and line number into a string, and appends it to the res string. If no matches are found, it sets the res string to indicate that no matches were found.

## Receiver

The Receiver class extends the common.Parent class and represents a receiver in the network communication system. It has the following attributes:

registry: An instance of the Registry class from the RMI (Remote Method Invocation) framework.

patternFinderRemote: A static attribute of type PatternFinderRemote used to access the remote methods provided by the sender.

The class has the following methods:

Receiver(String startingPort): The constructor that takes a starting port as a parameter. It calls the constructor of the superclass (Parent) and initializes the registry attribute.

startExecution(): A public method that represents the main execution logic of the receiver. It prints a message indicating that it is running in receiver mode and calls the connectToSocket() method.

connectToSocket(): A private method that establishes a connection with the middle server. It prompts the user to enter the IP address and port of the middle server, creates a socket connection with the middle server, and sets up input and output streams for communication.

The method handles the login process by reading login prompts from the middle server and sending the user's credentials.

After successful login, the method enters a loop where it prompts the user for input. If the user enters "1", the receiver requests the list of senders from the middle server and displays it. The user can then select a sender to connect to by entering the corresponding number.

If the user enters "2", the connection is closed, and the loop is terminated.

The method also handles the case where the user enters an invalid input.

connectToRMI(String connectionURL): A public method that connects to the RMI registry of a specific sender. It takes the connection URL (IP address and port) of the sender as a parameter.

The method uses the LocateRegistry class to get the RMI registry of the sender based on the provided connection URL.

It then looks up the remote object with the name MyConst.REGISTRY\_NAME in the registry, which corresponds to the PatternFinderRemote object exported by the sender.

The method enters a loop where it prompts the user to enter a string to search in the sender's data. If the user enters "Exit", the loop is terminated. Otherwise, the method calls the findPattern method on the patternFinderRemote object and prints the result.