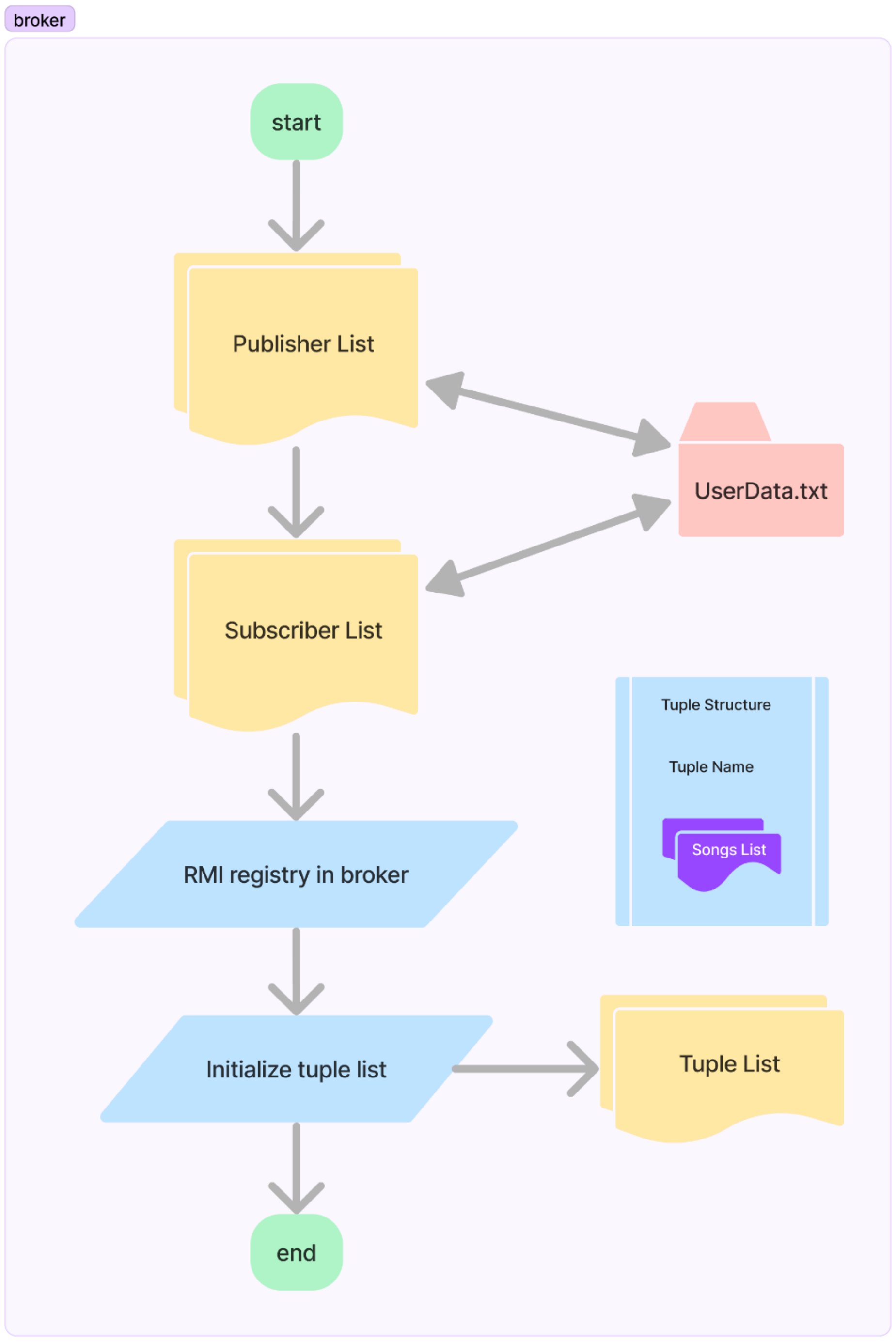
Advanced Operating System
Project-2

Submission by Kundanapreethi Surepally
(2249580)

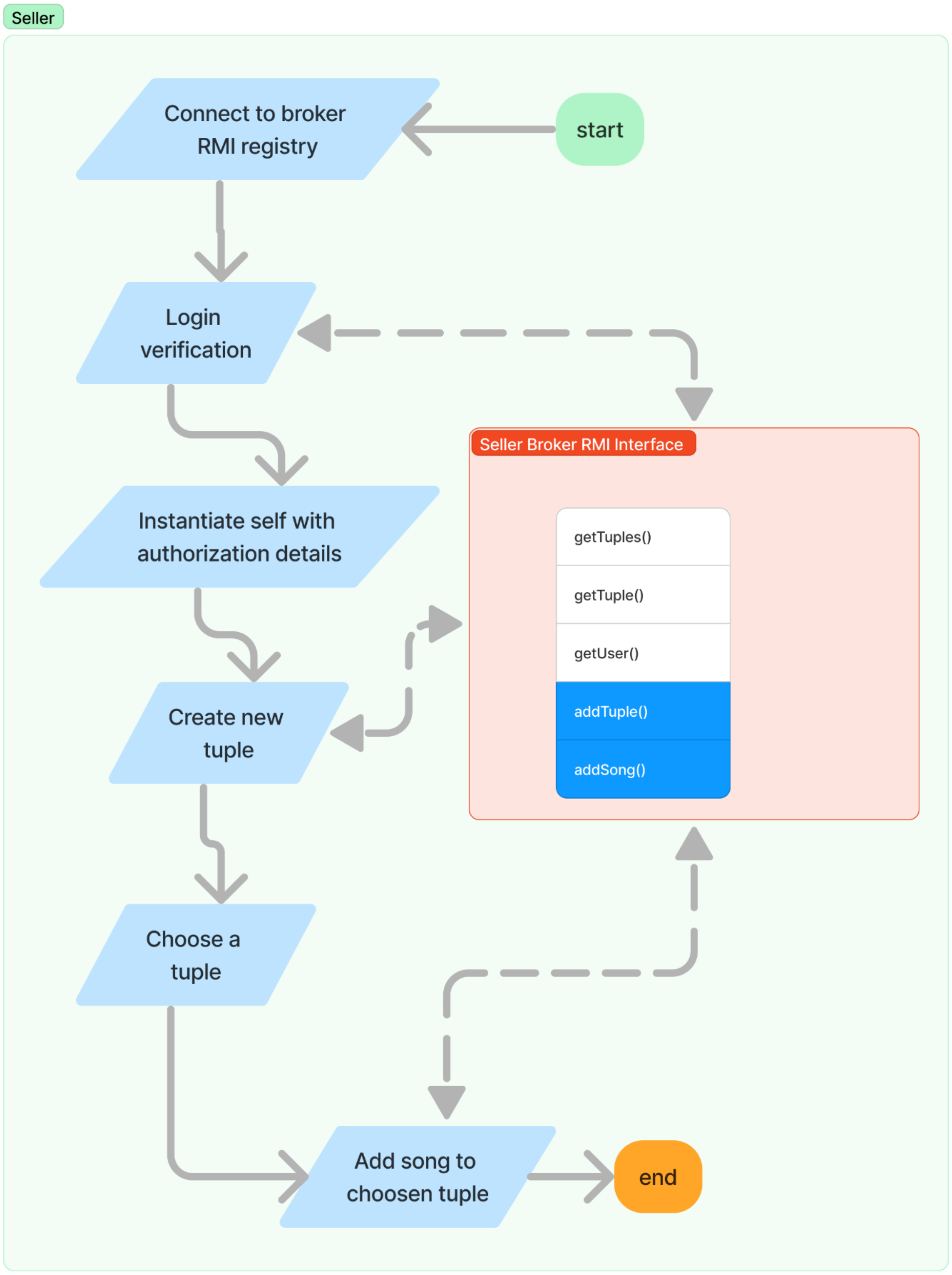
Table of Contents

Flow Chart:

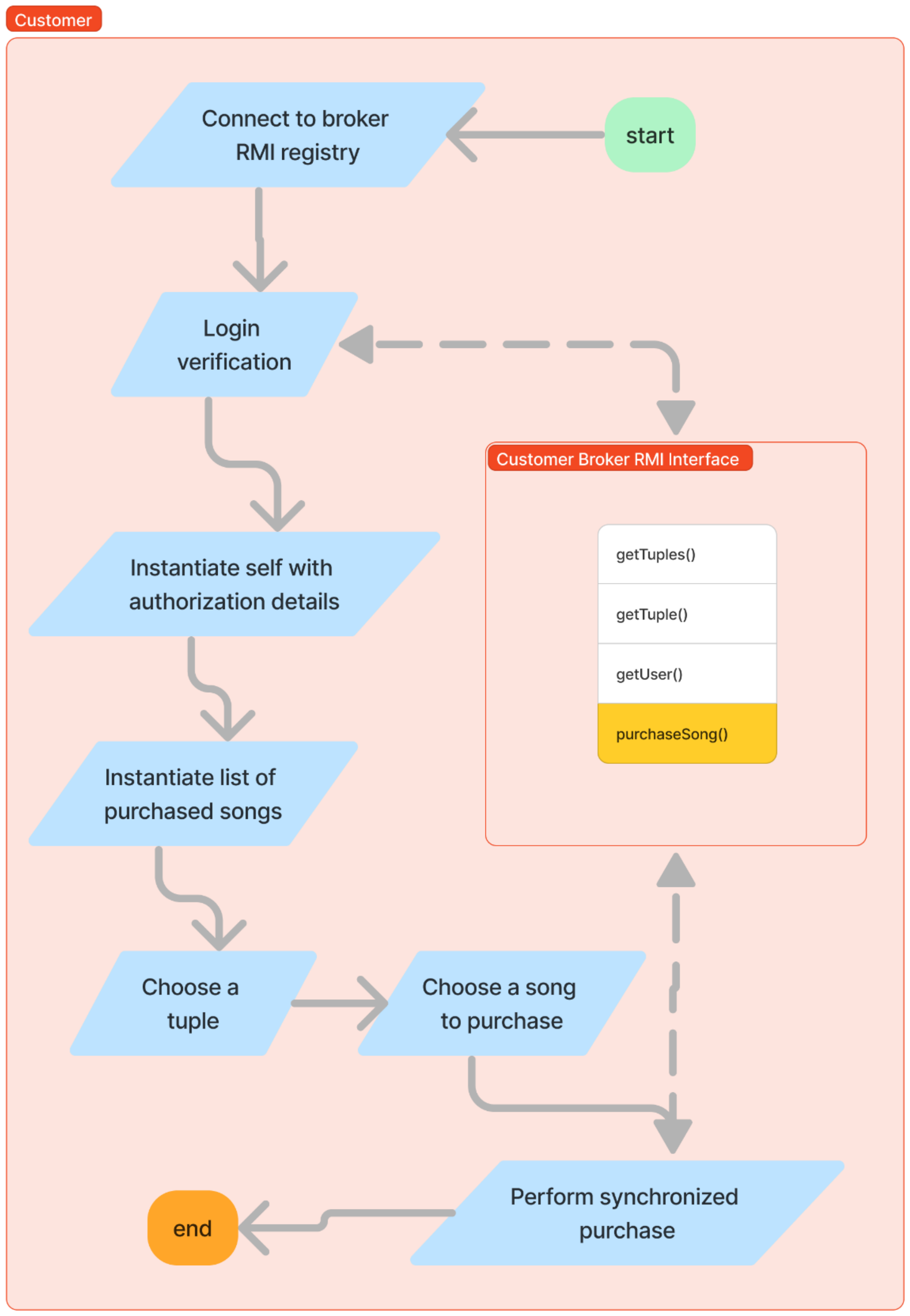
Broker



Seller


Seller
Connect to broker RMI registry
start
Login verification
Seller Broker RMI Interface
getTuples()
getTuple()
getUser()
addTuple()
addSong()
Instantiate self with authorization details
Create new tuple
Choose a tuple
Add song to choosen tuple
end

Customer

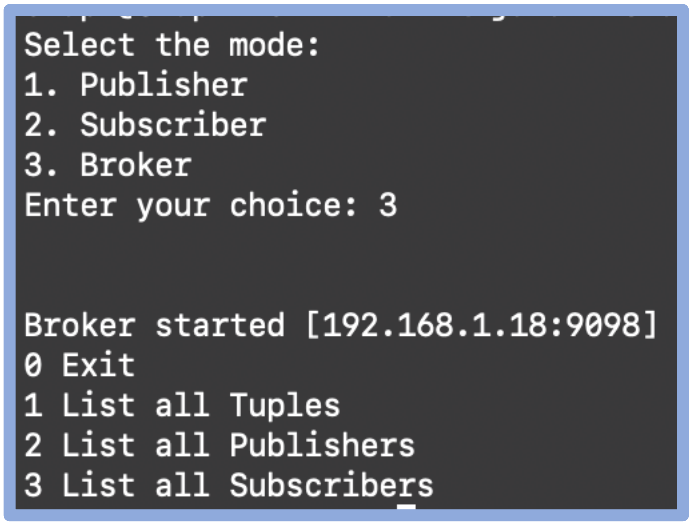

Customer
Connect to broker RMI registry
start
Login verification
Customer Broker RMI Interface
getTuples()
getTuple()
getUser()
purchaseSong()
Instantiate self with authorization details
Instantiate list of purchased songs
Choose a tuple
Choose a song to purchase
Perform synchronized purchase
end

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:



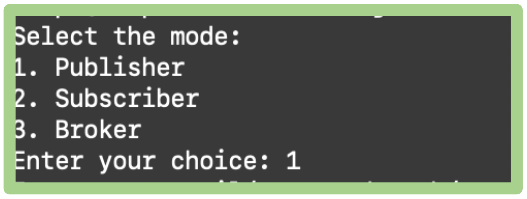
```
Select the mode:
1. Publisher
2. Subscriber
3. Broker
Enter your choice: 3


Broker started [192.168.1.18:9098]
0 Exit
1 List all Tuples
2 List all Publishers
3 List all Subscribers
```

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

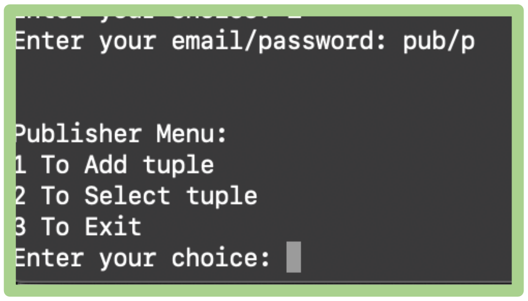
To start publisher run the same program in another system with ip address of the broker as a runtime argument and choose 1 in 1st menu

```
> java Driver com.pubsub.Driver 192.168.1.18:9098
```



```
Select the mode:
1. Publisher
2. Subscriber
3. Broker
Enter your choice: 1
```

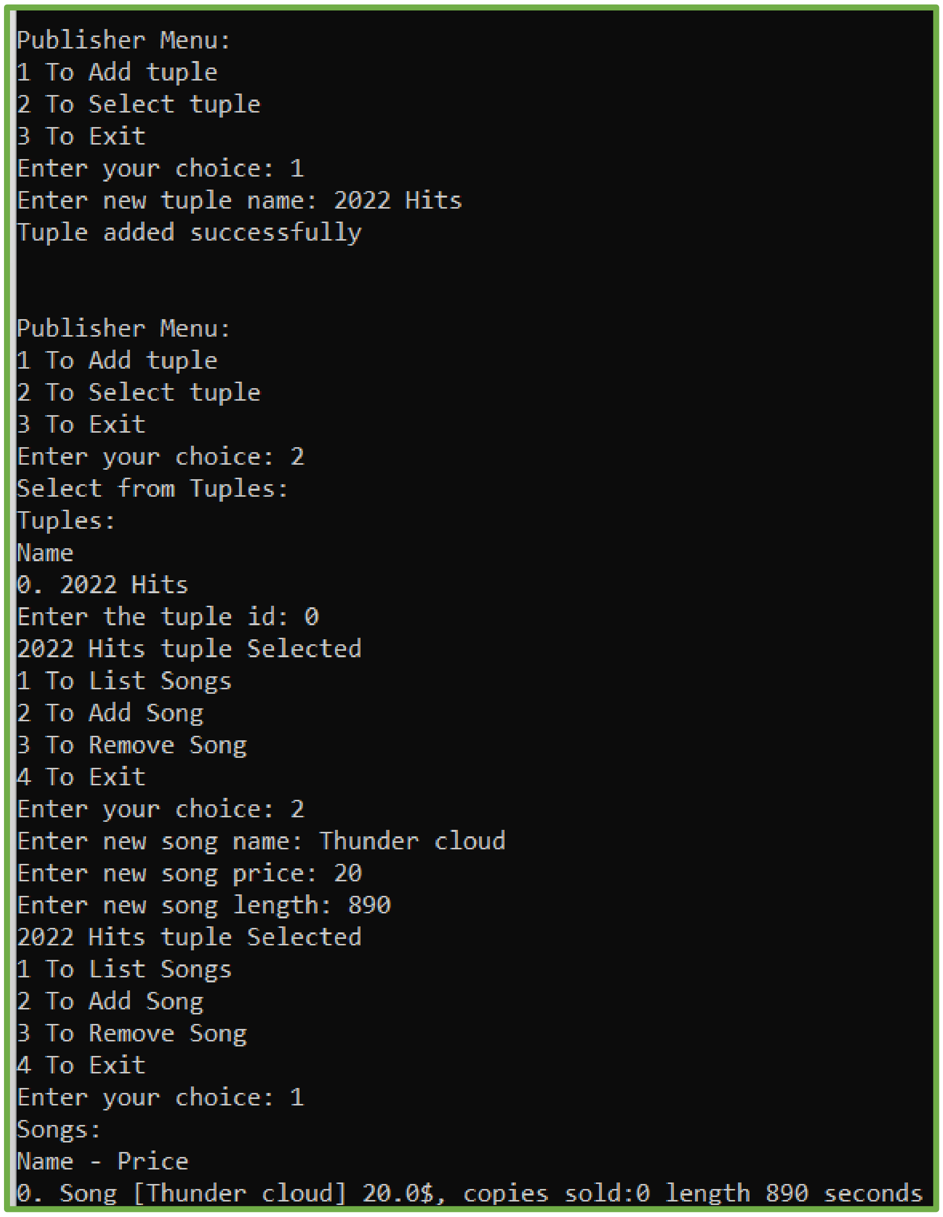
Now It will prompt to input username/password, on successful authentication, publisher will have access to tuple write and read rmi methods of broker



```
Enter your email/password: pub/p


Publisher Menu:
1 To Add tuple
2 To Select tuple
3 To Exit
Enter your choice: 
```

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

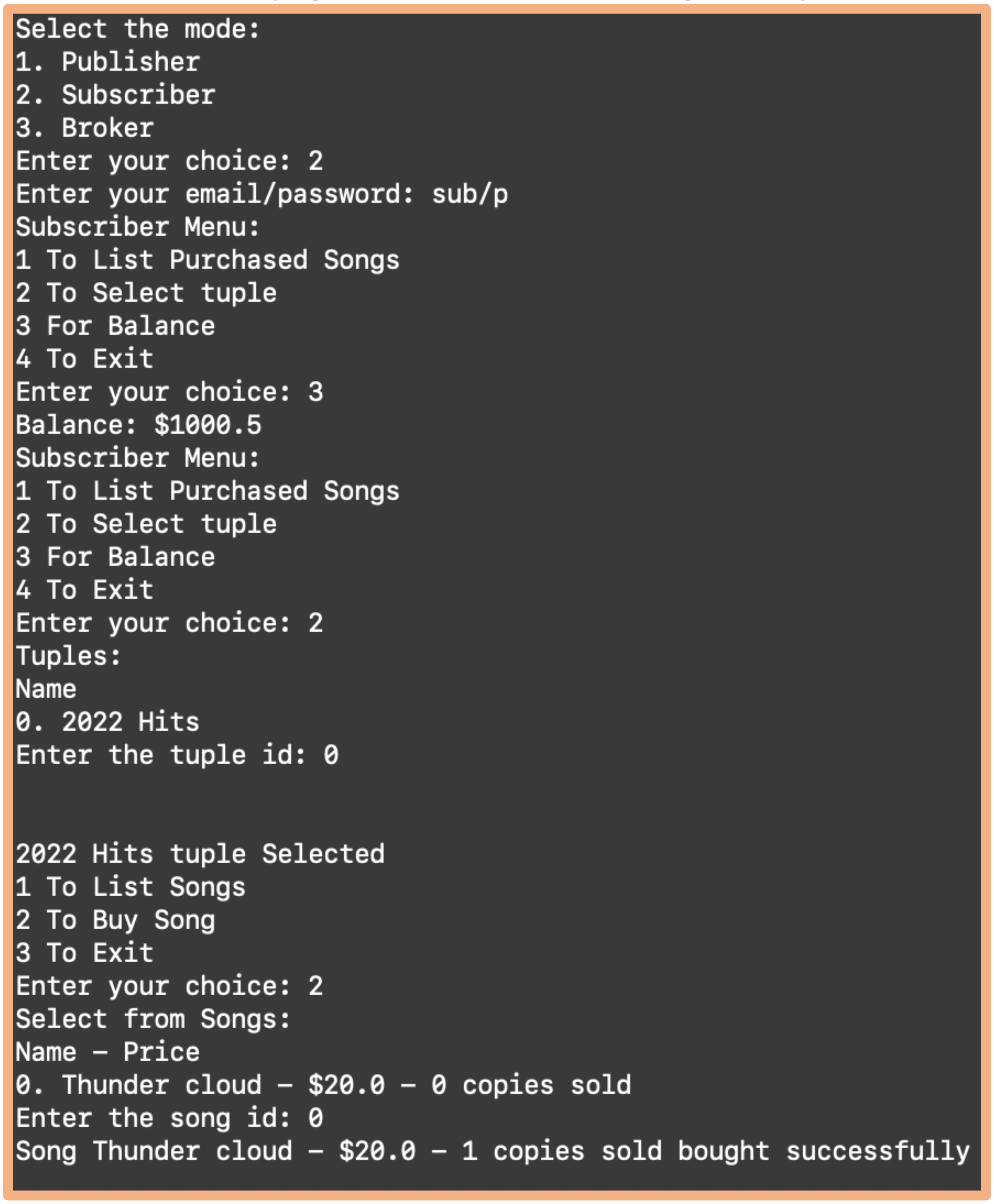


```
Publisher Menu:
1 To Add tuple
2 To Select tuple
3 To Exit
Enter your choice: 1
Enter new tuple name: 2022 Hits
Tuple added successfully


Publisher Menu:
1 To Add tuple
2 To Select tuple
3 To Exit
Enter your choice: 2
Select from Tuples:
Tuples:
Name
0. 2022 Hits
Enter the tuple id: 0
2022 Hits tuple Selected
1 To List Songs
2 To Add Song
3 To Remove Song
4 To Exit
Enter your choice: 2
Enter new song name: Thunder cloud
Enter new song price: 20
Enter new song length: 890
2022 Hits tuple Selected
1 To List Songs
2 To Add Song
3 To Remove Song
4 To Exit
Enter your choice: 1
Songs:
Name - Price
0. Song [Thunder cloud] 20.0$, copies sold:0 length 890 seconds
```

```
> java Driver com.pubsub.Driver 192.168.1.18:9098
```

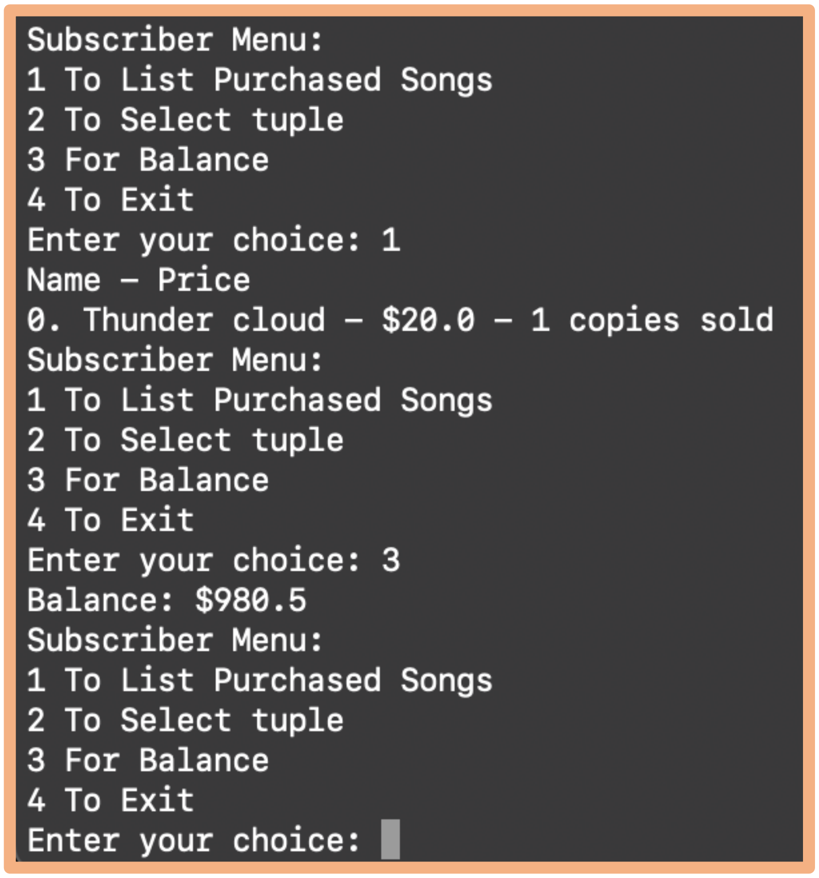
Now start the subscriber program with similar command line argument as publisher



```
Select the mode:
1. Publisher
2. Subscriber
3. Broker
Enter your choice: 2
Enter your email/password: sub/p
Subscriber Menu:
1 To List Purchased Songs
2 To Select tuple
3 For Balance
4 To Exit
Enter your choice: 3
Balance: $1000.5
Subscriber Menu:
1 To List Purchased Songs
2 To Select tuple
3 For Balance
4 To Exit
Enter your choice: 2
Tuples:
Name
0. 2022 Hits
Enter the tuple id: 0


2022 Hits tuple Selected
1 To List Songs
2 To Buy Song
3 To Exit
Enter your choice: 2
Select from Songs:
Name - Price
0. Thunder cloud - $20.0 - 0 copies sold
Enter the song id: 0
Song Thunder cloud - $20.0 - 1 copies sold bought successfully
```

To check the wallet balance:


```
Subscriber Menu:
1 To List Purchased Songs
2 To Select tuple
3 For Balance
4 To Exit
Enter your choice: 1
Name - Price
0. Thunder cloud - $20.0 - 1 copies sold
Subscriber Menu:
1 To List Purchased Songs
2 To Select tuple
3 For Balance
4 To Exit
Enter your choice: 3
Balance: $980.5
Subscriber Menu:
1 To List Purchased Songs
2 To Select tuple
3 For Balance
4 To Exit
Enter your choice: 
```

Classes:

Tuple

The Tuple class represents a model for a tuple object within the publisher-subscriber system. It encapsulates information about the tuple, such as its name and a list of associated songs.

1. The Tuple class is defined within the com.pubsub.model package.
2. The class implements the java.io.Serializable interface, which allows instances of the class to be serialized.
3. The class includes two instance variables:
4. name: A string representing the name of the tuple.
5. songs: A list of Song objects associated with the tuple.
6. The class has a constructor that takes the name of the tuple as an argument. It initializes the name variable and creates a new LinkedList to store the songs.
7. The class overrides the toString() method to provide a string representation of the Tuple object. The returned string includes the tuple's name and the string representation of each song in the songs list.

Song

1. The Song class is defined within the com.pubsub.model package.
2. The class implements the java.io.Serializable interface, which allows instances of the class to be serialized.
3. The class includes several instance variables:
4. name: A string representing the name of the song.
5. price: A double representing the price of the song.
6. copiesSold: An integer representing the number of copies sold for the song.
7. lengthInSeconds: A long integer representing the length of the song in seconds.
8. The class has a constructor that takes the name, price, and length of the song as arguments. It initializes the instance variables accordingly and sets copiesSold to 0.
9. The class overrides the toString() method to provide a string representation of the Song object. The returned string includes the song's name, price, number of copies sold, and length in seconds.

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

Broker

This class acts as a broker in a publisher-subscriber system. Here's a breakdown of the code:

1. The Broker class is defined within the com.pubsub.broker package.
2. The necessary imports are included at the beginning of the code.
3. The Broker class extends UnicastRemoteObject and implements the SellerBrokerInterface and CustomerBrokerInterface interfaces. These interfaces are not shown in the provided code snippet.
4. The class includes various instance variables:
    a. DATA_FILENAME: A constant string representing the path to the user data file.
    b. scanner: A Scanner object used for user input.
    c. tuples: A list of Tuple objects representing topics and associated songs.
    d. publishers: A list of User objects representing publishers.
    e. subscribers: A list of User objects representing subscribers.
    f. startingPort: An integer representing the starting port for the RMI registry.
5. The Broker class has a constructor that takes a Scanner object and a starting port as arguments. It initializes the instance variables and loads user data from a file using the loadUserData() method. It also starts the RMI registry by calling the startRegistry() method.
6. The startRegistry() method creates an RMI registry using the starting port and binds the Broker object to the registry with the name "Broker".
7. The loadUserData() method reads user data from a file (userData.txt) and populates the publishers and subscribers lists based on the data. Each line in the file represents a user and contains username, password, wallet balance, and a flag indicating whether the user is a publisher or subscriber.
8. The addTuple() method adds a Tuple object to the tuples list.
9. The getTuple() method retrieves a Tuple object from the tuples list based on the provided item name.
10. The getTuples() method returns the tuples list.
11. The startExecution() method represents the main functionality of the Broker class. It continuously prompts the user for input and performs actions based on the chosen option. The available options are:
    - 0: Exit the program.
    - 1: List all tuples (topics) and associated songs.
    - 2: List all publishers.
    - 3: List all subscribers.
12. The addSong() method adds a Song object to the songs list of a specific Tuple object based on the provided tuple name.
13. The saveUserData() method saves user data to the user data file. It takes a list of User objects and a boolean flag indicating whether to append the data or overwrite the file.
14. The purchaseSong() method allows a customer to purchase a song. It decrements the customer's wallet balance, increments the song's copies sold, and saves the updated user data to the file.

15. The getUser() method retrieves a User object based on the provided username and password. It searches through the publishers and subscribers lists to find a matching user.
16. The code includes exception handling for RemoteException, UnknownHostException, and IOException.

Seller

Seller class is a type of end user in the publisher-subscriber system. Here's a breakdown of the code:

1. The Seller class is defined within the com.pubsub.node.seller package.
2. The necessary imports are included at the beginning of the code.
3. The Seller class extends the EndUser class, which is not shown in the provided code snippet.
4. The Seller class has a constructor that takes a connection URL and a Scanner object as arguments. It calls the constructor of the EndUser class, passing the same arguments.
5. The startExecution() method represents the main functionality of the Seller class. It establishes a connection to the broker via RMI, retrieves the SellerBrokerInterface from the registry, and performs actions based on the user's input.
6. Inside the startExecution() method, a loop is used to repeatedly prompt the user for input until the choice is greater than or equal to 3.
7. The available options in the seller menu are:
   1: Add a new tuple.
   2: Select a tuple and perform actions on it (list songs, add a song).
   3: Exit the seller menu.
8. If the user chooses to add a new tuple (option 1), they are prompted to enter the tuple name, and the addTuple() method of the broker is called with a new Tuple object created using the provided name.
9. If the user chooses to select a tuple (option 2), they are presented with a list of tuples obtained from the broker using the getTuples() method. They can then choose a tuple by its ID and perform actions on it.
10. Inside the tuple menu, the available options are:
    1: List all songs in the selected tuple.
    2: Add a new song to the selected tuple.
    3: Exit the tuple menu.
11. If the user chooses to list songs (option 1), the songs of the selected tuple are retrieved from the broker using the getTuple() method and displayed.
12. If the user chooses to add a new song (option 2), they are prompted to enter the song name and price. Then, the addSong() method of the broker is called with the selected tuple name and a new Song object created using the provided name and price.
13. The code includes exception handling for any exception that might occur during the execution and prints the stack trace.

Customer

1. The Customer class is defined within the com.pubsub.node.customer package.
2. The necessary imports are included at the beginning of the code.
3. The Customer class extends the EndUser class, which is not shown in the provided code snippet.
4. The class includes an additional instance variable:
   songs: A list of Song objects representing the songs purchased by the customer.
5. The Customer class has a constructor that takes a connection URL and a Scanner object as arguments. It calls the constructor of the EndUser class, passing the same arguments, and initializes the songs list.
6. The startExecution() method represents the main functionality of the Customer class. It establishes a connection to the broker via RMI, retrieves the CustomerBrokerInterface from the registry, and performs actions based on the user's input.
7. Inside the startExecution() method, a loop is used to repeatedly prompt the user for input until the choice is greater than or equal to 4.
8. The available options in the customer menu are:
   1: List all purchased songs.
   2: Select a tuple and perform actions on it (list songs, buy a song).
   3: Check the wallet balance.
   4: Exit the customer menu.
9. If the user chooses to list purchased songs (option 1), the songs in the songs list are displayed using the printSongs() method.
10. If the user chooses to select a tuple (option 2), they are presented with a list of tuples obtained from the broker using the getTuples() method. They can then choose a tuple by its ID and perform actions on it.
11. Inside the tuple menu, the available options are:
    1: List all songs in the selected tuple.
    2: Buy a song from the selected tuple.
    3: Exit the tuple menu.
12. If the user chooses to list songs (option 1), the songs of the selected tuple are retrieved from the broker using the getTuple() method and displayed using the printSongs() method.
13. If the user chooses to buy a song (option 2), they are prompted to choose a song from the selected tuple and provide the song ID. The purchaseSong() method of the broker is called with the customer's username, selected tuple name, and song ID. If the purchase is successful, the customer's wallet balance is updated, and the purchased song is added to the songs list.
14. If the user chooses to check the wallet balance (option 3), the current wallet balance of the customer is displayed.
15. The code includes exception handling for NumberFormatException, RemoteException, and NotBoundException.