Problem Statement:

Given a Player class - an instance of which can communicate with other Players.

The requirements are as follows:

1. create 2 Player instances

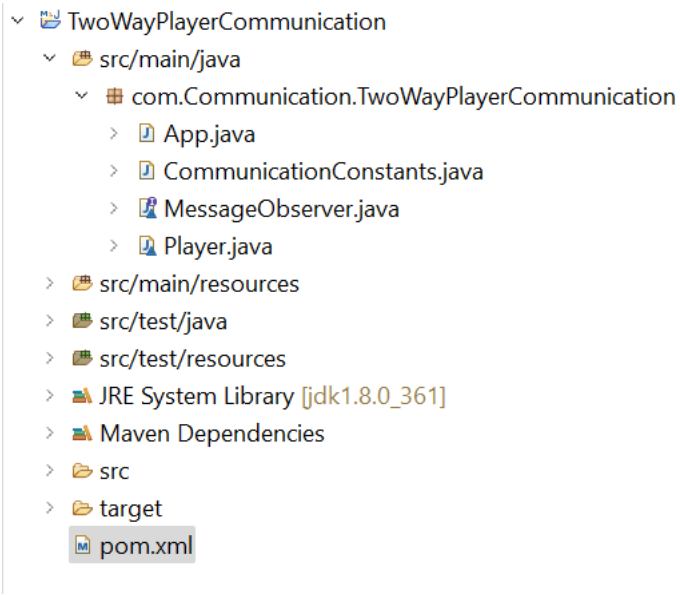
2. one of the players should send a message to second player (let's call this player "initiator")

3. when a player receives a message, it should reply with a message that contains the received message concatenated with the value of a counter holding the number of messages this player already sent.

4. finalize the program (gracefully) after the initiator sent 10 messages and received back 10 messages (stop condition)

5. both players should run in the same java process (strong requirement)

Project Structure:



App.java (Main class)

1. This is the entry point of the program.
2. Creates two instances of the `Player` class, `player1` and `player2`.
3. Associates each player with the other player.
4. Starts two threads, `thread1` and `thread2`, each corresponding to a player.
5. Randomly chooses one player to initiate communication.
6. Waits for the player threads to finish execution using the `join` method. The join method is used to ensure that the main thread waits for the two player threads (thread1 and thread2) to finish their execution before the program terminates. This is important to ensure proper synchronization and allows players to complete the communication.

Player.java

1. This class is used to represent a player participating in the communication.
2. Used Builder Design Pattern to create instances of 'player' class.

3. Maintains ‘BlockingQueue’ that is named Message Queue in order to hold messages.
4. Implements the `MessageObserver` interface to receive and process messages.
5. sendMessage method adds messages to the player's message queue.
6. notify method, from the `MessageObserver` interface, processes received messages.
7. receiveMessage method handles processing and forwarding of received messages.

MessageObserver.java

1. The `MessageObserver` interface with a single method, `notify`, which allows an object to receive and handle messages.

CommunicationConstants.java

1. Included separate constant class to avoid hard coding.

Explanation of Design Pattern used:

Builder Design Pattern:

1. Clear way to create `Player` instances.
2. Utilized in the `Player` class using the inner `Builder` class.
3. Separates the construction of a complex object (`Player`) from its representation.
4. Advantages: Improved code readability, reduced parameter complexity.

Observer Design Pattern:

1. Implemented via the `MessageObserver` interface.
2. Enables one-to-many dependency relationships between objects.
3. Notifies dependent objects of changes, promoting loose coupling.
4. In the code, `Player` instances act as observers, receiving and responding to messages from other players.

Final Output:

Both players in same java Process.

```
D:\Workspace\360T_Problem\TwoWayPlayerCommunication>java -cp ./target/TwoWayPlayerCommunication-0.0.1-SNAPSHOT.jar com.Communication.TwoWayPlayerCommunication.App
Player2 received message: Hello, sending response: Hello 1
Player1 received message: Hello 1, sending response: Hello 1 1
Player2 received message: Hello 1 1, sending response: Hello 1 1 2
Player1 received message: Hello 1 1 2, sending response: Hello 1 1 2 2
Player2 received message: Hello 1 1 2 2, sending response: Hello 1 1 2 2 3
Player1 received message: Hello 1 1 2 2 3, sending response: Hello 1 1 2 2 3 3
Player2 received message: Hello 1 1 2 2 3 3, sending response: Hello 1 1 2 2 3 3 4
Player1 received message: Hello 1 1 2 2 3 3 4, sending response: Hello 1 1 2 2 3 3 4 4
Player2 received message: Hello 1 1 2 2 3 3 4 4, sending response: Hello 1 1 2 2 3 3 4 4 5
Player1 received message: Hello 1 1 2 2 3 3 4 4 5, sending response: Hello 1 1 2 2 3 3 4 4 5 5
Player2 received message: Hello 1 1 2 2 3 3 4 4 5 5, sending response: Hello 1 1 2 2 3 3 4 4 5 5 6
Player1 received message: Hello 1 1 2 2 3 3 4 4 5 5 6, sending response: Hello 1 1 2 2 3 3 4 4 5 5 6 6
Player2 received message: Hello 1 1 2 2 3 3 4 4 5 5 6 6, sending response: Hello 1 1 2 2 3 3 4 4 5 5 6 6 7
Player1 received message: Hello 1 1 2 2 3 3 4 4 5 5 6 6 7, sending response: Hello 1 1 2 2 3 3 4 4 5 5 6 6 7 7
Player2 received message: Hello 1 1 2 2 3 3 4 4 5 5 6 6 7 7, sending response: Hello 1 1 2 2 3 3 4 4 5 5 6 6 7 7 8
Player1 received message: Hello 1 1 2 2 3 3 4 4 5 5 6 6 7 7 8, sending response: Hello 1 1 2 2 3 3 4 4 5 5 6 6 7 7 8 8
Player2 received message: Hello 1 1 2 2 3 3 4 4 5 5 6 6 7 7 8 8, sending response: Hello 1 1 2 2 3 3 4 4 5 5 6 6 7 7 8 8 9
Player1 received message: Hello 1 1 2 2 3 3 4 4 5 5 6 6 7 7 8 8 9, sending response: Hello 1 1 2 2 3 3 4 4 5 5 6 6 7 7 8 8 9 8
Player2 received message: Hello 1 1 2 2 3 3 4 4 5 5 6 6 7 7 8 8 9 8, sending response: Hello 1 1 2 2 3 3 4 4 5 5 6 6 7 7 8 8 9 8 10
Player1 received message: Hello 1 1 2 2 3 3 4 4 5 5 6 6 7 7 8 8 9 8 10, sending response: Hello 1 1 2 2 3 3 4 4 5 5 6 6 7 7 8 8 9 8 10 10

D:\Workspace\360T_Problem\TwoWayPlayerCommunication>
```