PROBLEM DEFINITION

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 the 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.
5. Players should run in separate Java processes.

PROGRAM SUMMARY

- This is a Java program demonstrating inter-process communication by using socket programming.
- It simulates the communication between 2 distinct players via 2 distinct processes.
- There are 2 subclasses of the base Player class corresponding to the initiator and receiver players: InitiatorPlayer and ReceiverPlayer.
- There are 2 classes that have 2 distinct main methods corresponding to the initiator and receiver processes to manage inter-process communication between the initiator player and receiver player: InitiatorProgram and ReceiverProgram.
- The initiator process is responsible for starting and finishing the communication. It starts the communication with a **"Hello 0"** message. Once it sends and receives 10 messages back, it sends a **"FINISH"** message to the receiver process and terminates. It uses an instance of the InitiatorPlayer that maintains the initiator side of the data.
- The receiver process is responsible for responding back to the messages of the initiator process. Once it gets a **"FINISH"** message from the initiator process, it terminates. It uses an instance of the ReceiverPlayer that maintains the receiver side of the data.
- Except for the first and last messages that the initiator process sends **("Hello 0" and "FINISH")**, all messages are created by adding the message count that the current process already sent to the received message from the opposite process (excluding the message that is being sent in that cycle).
- The communication is maintained via a socket connection established between the initiator and receiver processes.

CLASS SUMMARY

There are 5 classes in the program. Their summaries are given below:

Player

- It is an abstract class that constructs a base for the InitiatorPlayer and ReceiverPlayer classes with common attributes and methods.
- It has a Socket instance to maintain the communication between processes. Both sides of the communication must have a Socket instance because the messages are sent and received via this channel.

InitiatorPlayer

- It extends the Player class.
- The receivedMessageCount variable was added to this class. It does not occur in the ReceiverPlayer class because the initiator process controls the termination of the processes via InitiatorPlayer class by using this variable. Once this variable reaches 10, it means that the initiator process sent and received 10 messages. So, the initiator process terminates and a **"FINISH"** message is sent to the receiver process.

ReceiverPlayer

- It extends the Player class.
- A ServerSocket instance was added to this class. It does not occur in the InitiatorPlayer class because the receiver process listens to the port via ReceiverPlayer class by using this instance. Once a request comes to the port from the initiator process, a socket connection is established between the initiator and receiver processes via the Socket instance in the base Player class.

InitiatorProgram

- It corresponds to the initiator process and has a main method.
- It instantiates all necessary objects for the socket connection.
- This class is responsible for the operations summarized in the program summary.

ReceiverProgram

- It corresponds to the receiver process and has a main method.
- It instantiates all necessary objects for the socket connection.
- This class is responsible for the operations summarized in the program summary.

NOTES

- This project was developed with Java 1.8.
- This is a Maven project created in Eclipse.
- There are 3 **".sh"** files in the root directory.
  - First, double-click on the **"compile.sh"**. It compiles all the necessary files.
  - Second, double-click on the **"start_receiver.sh"**. It runs the ReceiverProgram and waits for the execution of the InitiatorProgram.
  - Third, double-click on the **"start_initiator.sh"**. It runs the InitiatorProgram and the communication starts.