Player Communication project by Sanzhar Aubakir for 360T

Project Overview:

For this project I have used Gradle as a build tool and JUnit for tests. As it was mentioned in the task I have not used any frameworks, but only core java.

I have implemented both #5 and #7 requirements: so if a user enters argument SINGLE - then whole process would work on SAME process, if user enters SEPARATE - then whole process would work on Separate processes.

Implementation:

App class is the main application class.

ArgumentParser class parses the command line arguments and retrieves whether application should run on SINGLE process or on SEPARATE processes.

Player interface defines two main methods that Player instances must realize/

AbstractPlayer class is defining how Player instance should behave, it has player name, player type (INITIATOR or RESPONDENT, INITIATOR is the first player to send the message), receivedCounter and sentCounter (to keep track of number of received and sent messages). AbstractPlayer extends Runnable interface and has run method, which is realizes basic functionality: player receives a message, waits for 1 second (imitates reading and typing) and sends the message + sentCouner back. It stops receiving once it has seen 10 messages and stops sending once it has send 10 messages. AbstractPlayer class also extends Player interface, which has receive and send messages. Implementation of send and receive methods are done by children classes of AbstractPlayer, making the architecture very flexible: communication method is defined by child classes and main functionality remains the same.

Constants class contains all required constans.

Message class is an object containing contents of the message and sender name.

GameType is enum for SINGLE or SEPARATE process implementation.

PlayerType is enum for INITIATOR or RESPONDENT player type.

Same Process Implementation:

ThreadPlayer class is a child of AbstractPlayer class which realizes send and receive methods by thread communication. I have created two instances of a ThreadPlayer running on two separate threads. For communication those threads use LinkedBlockingDeque which is thread-safe. When player wants to send the message, message is placed on receiving player`s queue.

Separate Process Implementation:

ServerConnection class extends Thread class, and is created for every client connection (INITIATOR and RESPONDENT), it keeps the connection, receives and sends messages through socket. Once ServerConnection receives the message from the client it uses list of ServerConnections to send the message to another client.

SocketPlayer class is a child of AbstractPlayer which realizes send and receive methods using socket communication through server.

For separate process I have decided to implement multi-client chat server, so there is a server thread which accepts client connection requests and creates a Thread which maintains socket communication and is called ServerConnection. Player instances Initiator and Respondent are working on SocketPlayer class which extends AbstractPlayer class. SocketPlayer class have a socket connection with server and sends and receives messages using BufferedReader and BufferedWriter. When both Initiator and Respondent SocketPlayers are connected to server and have their correspondent ServerConnection instances, server thread no longer accepts connection requests and Initiator sends the first message.

I have read javadoc and tests for the application which may be seen using following commands:

Building, Running, Javadoc, Tests:

Build the project:

gradle jar

Run the project:

Same java process:

./single_process.sh (Need to give permission (chmod))

Separate java process:

./separate_process.sh (Need to give permission (chmod))

Run tests:

gradle test

test results may be seen in build/reports/tests/test folder index.html

Generate javadoc:

gradle javadoc

generates javadoc, may be seen in build/docs/javadoc folder index.html