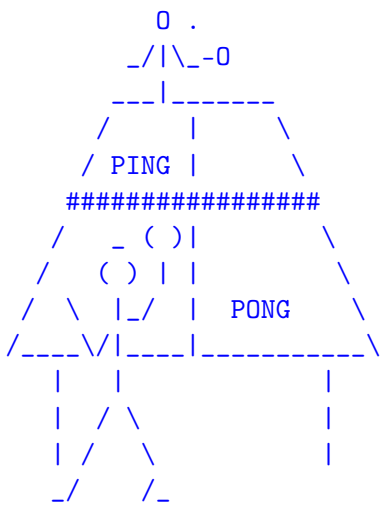
CS 455-555: Distributed Systems

Homework 1 (50 points)

Due Date –On Canvas–

Objective

Simulate a ping pong game using a server and a client.



Setup

Make sure to accept the GitHub invitation for an individual repository for hw and exams (this should have been done in a previous assignment). Then copy and commit the starter files for hw1 from the `CS455-resources` repository in the folder `projects/hw1`.

Implementation

Designate one player as the “server" (the one with a `ServerSocket`) and one player as a “client" (the one with a `Socket` that connects to the server). This role would be specified via a command line argument.

```
java PingPong
Usage:  java PingPong <client|server> <serverHost> <port#>
```

Note that if the role is server, then the *serverHost* would always be `localhost`.

For example, we would start the server in a console as follows:

```
[amit@localhost hw1(master)]$ java PingPong server localhost 5005
```

It would then block waiting for a client. Meanwhile, we can start the client in another console as follows:

```
[amit@localhost hw1(master)]$ java PingPong client localhost 5005
```

Representing a ping pong ball. You can simply use an empty object to simulate a ping pong ball.

```
public class Ball extends Object implements Serializable {}
```

Assume that the client always starts with the ball and the server always starts first and waits for the client to connect.

Version 1: Fixed playing mode

The server and client are both single threaded and can be in the same class. Fix the client to always start the game with a "ping." In this case, the server always plays the "pong" part.

Version 2: Either server or client can start the game

Starting the game. The server and client participate in a distributed coin toss to choose who gets to start the game. The distributed coin toss is done by each process sending an arbitrary random number to the other process. The process that has the smaller number gets to start the game (that is, be the "ping"). In case of a tie, the processes repeat this protocol until the tie is broken. So there is an infinitesimal probability that the coin toss may go on forever (this is an example of how distributed algorithms are often different and fun!)

Playing the game. The process that gets to be "ping" starts the game. In case of the server, it would still need to get the ball from the client before initiating a "ping." The game continues indefinitely unless the client or the server dies (for example, with a `Ctrl-c` or a `kill` command). In this case, it is acceptable for the other server/client to also die with an exception.

Version 3: Multithreaded Server, Singlethreaded client

Modify the server so that it can play ping pong with multiple clients. It should create a thread for each client it connects with so that it play in parallel. Note that the server still does a distributed coin toss with each client as before. The server assigns a new game number to each game it starts (starting from 1) and prints the game number to the console. Game numbers keep on increasing linearly. We will have to synchronize the code that uses the variable to keep track of the game number.

It is very helpful to design and write the protocol more formally. In fact, for designing distributed applications the protocol design is often the crucial part. Give it a try yourself and then check out the protocol provided here: Ping Pong v3 protocol.

Output to console: Have reasonable output that shows the distributed coin toss, game numbers and then the actual ping/pong output representing the game. See your git repo for a sample screenshot.

Version 4: Multithreaded Servents (Extra credit)

We will make each player be a servent. A **servent** is a process that is both a server and a client (**ser**ver + cli**ent**). Each process now has a `ServerSocket` and a `Socket` to connect to each other. Each process has

two threads: one to send and one to receive. This is an extension of Version 2. If you attempt this part, place this in a separate subfolder under `hw1` folder and let me know in your `README.md` file.

Required Files

- Please use the separate folder named `hw1` for this homework in your GitHub classroom individual repo.
- Please use the class name `PingPong` for the main class for your program. Include a brief `README.md` that contains your name and instructions for me to run the program. Use the template file `README-hw-template.md` found in `CS455-resources/projects` folder.
- You only need to submit Version 3 of your code. The multiple versions specified above are to guide your development.
- If you are submitting Version 4 for extra credit please put that in a subfolder named `ec` inside the `hw1` folder.

1 AI Usage

Describe any use of AI tools (e.g., ChatGPT, GitHub Copilot, etc.) in completing this homework. Please note that it is acceptable to use AI tools to ask questions and help you debug code, but you must not use AI tools to generate code or complete homework for you. The homework is to help you solidify concepts you are learning in class, and using AI to generate code defeats that purpose.

Submitting the Homework

Homework is individual so we will use your individual github classroom repository. To obtain credit, submit your homework through classroom github as described below.

Change to the directory called hw1 in your classroom individual repo and place all required files in that folder. Then do the following steps (on your master branch):

- git add [The appropriate files]
- git commit -m "Homework hw1 complete"
- git push origin master