PUCRS

Pontifícia Universidade Católica
do Rio Grande do Sul

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL

ESCOLA POLITÉCNICA

CURSO DE BACHARELADO EM ENGENHARIA DE SOFTWARE

GABRIEL BILHAR,
FERNANDO ELGER,
FELIPE BORDIGNON

PROGRAMAÇÃO DISTRIBUÍDA

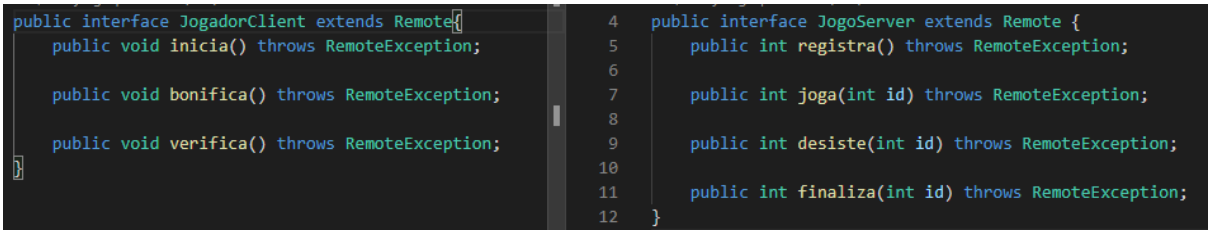
Trabalho 1

Porto Alegre

2020

1. Organização do código

Interfaces para comunicação



```java
public interface JogadorClient extends Remote{
    public void inicia() throws RemoteException;

    public void bonifica() throws RemoteException;

    public void verifica() throws RemoteException;
}
```

```java
public interface JogoServer extends Remote {
    public int registra() throws RemoteException;

    public int joga(int id) throws RemoteException;

    public int desiste(int id) throws RemoteException;

    public int finaliza(int id) throws RemoteException;
}
```

Início da execução

Tanto para o *client* quanto o *server*, o primeiro passo da execução é verificar se o registro do Java RMI já está criado e alocado na porta 52369:

```java
private static final int port = 52369;
```

```java
try {
    System.setProperty("java.rmi.server.hostname", args[0]);
    LocateRegistry.createRegistry(port);
    System.out.println("java RMI registry created.");
} catch (RemoteException e) {
    System.out.println("java RMI registry already exists.");
}
```

Client

Cada instância do client representa um jogador que irá se conectar ao servidor. Primeiramente se cria o registro do Java RMI definindo o ip do servidor e a porta em que será feita a comunicação, que por sua vez é a mesma porta do servidor. O *bind* do client é realizado com um endereço RMI formado pelo ip do jogador, sua porta e seu identificador. O que diferencia um client do outro é seu identificador, que contém uma variável volátil que vai sendo incrementada para cada jogador. Foi criado um tratamento para caso o valor da variável volátil já tiver sido usada em outro jogador. Se isso acontecer, o client tenta fazer o *bind* novamente com outro valor como identificador. Após este procedimento, é necessário fazer o *lookup* com o servidor com um endereço RMI composto pelo ip do servidor e sua porta.

Após o client fazer a conexão com o servidor, o jogador se registra no jogo. O método registra() realiza um *callback* para o client informando o id do jogador, que por sua vez é necessário para realizar as jogadas, para desistir ou para finalizar o jogo (métodos da interface do server). Como o jogo necessita de todos os jogadores registrados para iniciar, foi criado um *loop* para o client aguardar o início do jogo.

```
while (!gameStarted) {
  Thread.sleep(1000);
}
```

A variável *gameStarted* é um booleano que começa com valor false. Quando o servidor chama o método inicia da interface do cliente, o valor essa será atribuído o valor true para essa variável, para então sair do *loop* e começar a efetuar suas jogadas.

```
  int pontuacao = 0;
  for (int i = 0; i < Integer.parseInt(args[2]); i++) {
    pontuacao += jogoServer.joga(idJogador);
    System.out.println(pontuacao + " Njogada " + i);
  }
  jogoServer.finaliza(idJogador);
  System.exit(1);
}
```

Foi criado um *loop* para iterar em cima da quantidade de jogadas estabelecidas do jogador em questão para chamar o método joga() da interface do server. A pontuação do jogador recebida através de um *callback* do server é somada com a quantidade total de pontos. Após a finalização de todas as jogadas, o jogador chama o método finaliza() da interface do server e termina a execução.

Todos os métodos da interface do client printam na terminal quando o jogo inicia, quando o jogador é bonificado e quando o server realiza a verificação da conexão com o client.

Server

Ao iniciar o servidor, o primeiro passo executado será adicionar o endpoint para o *server* no registro do Java RMI.

Capturamos o valor recebido por linha de comando que irá definir quantos jogadores vão se conectar para jogar, em seguida, executamos um laço de repetição que será executado a cada segundo, a fim de verificar se todos os jogadores já estão conectados e prontos para jogar. Assim que essa condição for verdadeira, o servidor irá iterar pela lista de *hosts* (dados capturados no momento em que o *client* se registra) a fim de executar o método inicia() em cada *client*, para que comecem a jogar.

```
int numeroJogadores = Integer.parseInt(args[1]);

while (true) {
    Thread.sleep(1000);
    if (numeroJogadores == hosts.size()) {

        for (int i = 0; i < hosts.size(); i++) {
            String connectLocation = hosts.get(i);

            JogadorClient jogadorClient = null;
            try {
                System.out.println("Connecting to client at " + connectLocation);
                jogadorClient = (JogadorClient) Naming.lookup(connectLocation);
                jogadorClient.inicia();
            } catch (Exception ex) {
                System.err.println("Failed to call inicia() method on client");
                ex.printStackTrace();
            }
        }
        break;
    }
}
```

Logo em seguida o servidor irá iniciar a sua rotina até que todos os *clients* se desconectem, verificando a cada 5 segundos se o jogador ainda está ativo através de uma chamada remota para o método verifica(), também irá verificar se o jogador recebeu alguma bonificação.

```
while (true) {
    if(hosts.isEmpty()){
        break;
    }

    for (int i = 0; i < hosts.size(); i++) {
        if (numeroJogadores == hosts.size()) {
            JogadorClient jogadorClient = null;
            try {
                String connectLocation = hosts.get(i);
                if(connectLocation == null){
                    continue;
                }
                System.out.println("Connecting to client at " + connectLocation);
                jogadorClient = (JogadorClient) Naming.lookup(connectLocation);
            } catch (Exception ex) {
                System.err.println("Failed to connect to client");
                ex.printStackTrace();
            }

            if (seBonificou) {
                jogadorClient.bonifica();
                seBonificou = false;
            }

            try {
                jogadorClient.verifica();
            } catch (RemoteException ex) {
                hosts.remove(i);
                numeroJogadores--;
            }

        }
    }

    Thread.sleep(5000);
}
```

No método joga(), criamos uma chance de 3% de o jogador receber uma bonificação, também é gerado aleatoriamente um intervalo de 250 até 950 milissegundos para simular que ele está jogando, o método retorna para o jogador

qual a sua atual pontuação.

```java
@Override
public int joga(int id) throws RemoteException {
    System.out.println(String.format("Jogador ID %d vai jogar....", id));
    Random gerador = new Random();
    int bonifica = gerador.nextInt(99);

    if (bonifica <= 2) {
        seBonificou = true;
        playerScores.put(id, playerScores.get(id) + 6);
    } else {
        playerScores.put(id, playerScores.get(id) + 5);
    }
    int playTime = ThreadLocalRandom.current().nextInt(250, 950);

    System.out.println("Jogada vai levar " + playTime + "ms");

    try {
        Thread.sleep(playTime);
    } catch (InterruptedException ex) {
        ex.printStackTrace();
    }

    return playerScores.get(id);
}
```

2. Utilização do programa

Para a inicializar o funcionamento do programa, primeiro começar executar a classe JogoServerImpl, passando como parâmetros o ip servidor e a quantidade de jogadores que irão participar do jogo, o programa não será inicializado até chegar na mesma quantidade de jogadores que foi passado como parâmetro:

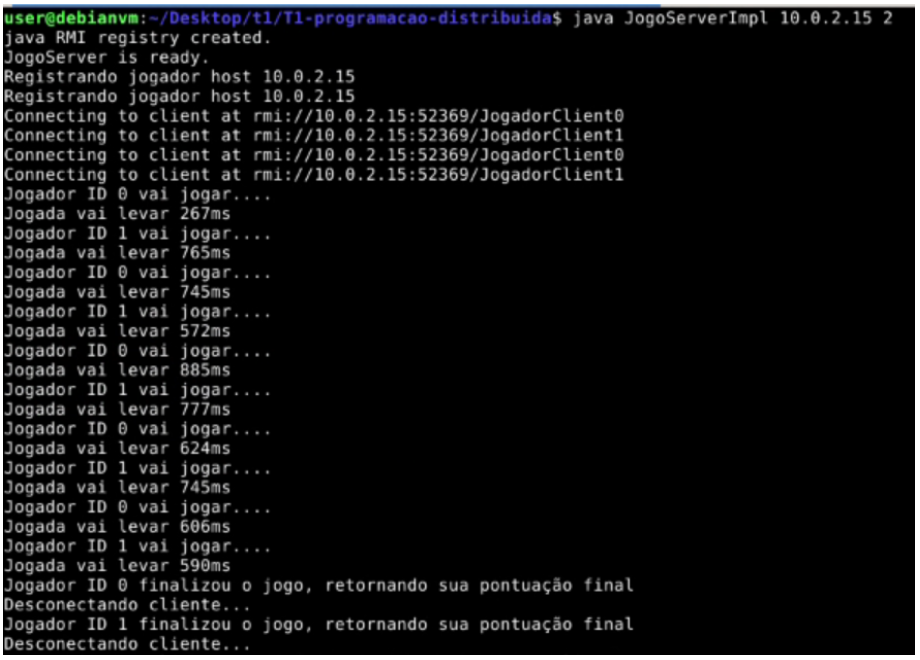
ex: java JogoServerImpl 10.0.2.15 2

Para parte dos jogadores, executar a classe JogadorClientImpl, passando como parâmetro ip do servidor que irá se juntar, o próprio ip e a quantidade de jogadas que irá executar.

ex: java JogadorClientImpl 10.0.2.15 2 10.0.2.15 2 10

3. Demonstração da implementação

```
user@debianvm:~/Desktop/t1/T1-programacao-distribuida$ java JogadorClientImpl 10.0.2.15 10.0.2.15 5
java RMI registry already exists.
JogadorClient is ready.
Connecting to server at : rmi://10.0.2.15:52369/JogoServer
JogoServer is ready.
Calling server to register player1
Player registered
Called back.
The game has started!
Called back.
Server ping test to verify player!
5 Njogada 0
15 Njogada 1
30 Njogada 2
50 Njogada 3
75 Njogada 4
user@debianvm:~/Desktop/t1/T1-programacao-distribuida$
```



Quando server for inicializado ele aguardará até que todos os jogadores se conectem no mesmo, quando o jogo iniciar, o server irá retornar qual jogador está jogando e em quanto tempo de *delay* terá entre as jogadas, cada jogada é aleatorizada entre cada jogador, há uma pequena chance do jogador receber uma bonificação em sua jogada e uma menor em desistir durante uma jogada, depois que um jogador terminar todas suas jogadas ele irá encerrar sua conexão com servidor, e após todos jogadores encerrarem todas suas jogadas irá encerrar o servidor.

Quanto aos jogadores, quando eles conectarem ao servidor, irá retornar em qual servidor ele está conectado e qual seu nome está registrado no mesmo, cada jogada que jogador fizer irá retornar qual sua pontuação e em qual jogada ele está, a pontuação vai aumentando de cinco em cinco pontos, caso ele tiver a sorte de ser bonificado durante a jogada, ele receberá um bônus de mais um ponto, ele irá jogando até acabar suas jogadas ou ter azar de ter chance de desistir do jogo durante umas das jogadas, assim encerrando sua conexão com servidor.