Programação Distribuída

Java RMI - T1

Giuseppe Menti

Instituto de Informática –Pontifícia Unidade Católica do Rio Grande do Sul (PUCRS)
Caixa Postal – 90.619-900– Porto Alegre – RS – Brazil

{giuseppe-menti@acad.pucrs.br}

Resumo. *O trabalho tem como objetivo exercitar conceitos relativos à implementação de serviços utilizando Java RMI. Consiste na simulação de uma comunicação entre jogadores e servidor em um jogo online com X jogadores em uma sala.*

1. Informações Gerais

O projeto foi desenvolvido utilizando a linguagem Java e a biblioteca java.rmi para realizar a comunicação entre os serviços, trata-se de um mecanismo para realizar a chamada de funções em diferentes máquinas utilizando java.]

2. Objetivo do trabalho

O objetivo do trabalho consiste em criar uma aplicação utilizando Java RMI para registrar usuários (jogadores) em um servidor remoto de jogos. Ao registrar um jogador no servidor, recebe-se um identificador único (um número inteiro). Este identificador único deve ser utilizado pelos jogadores como parâmetro em três outros métodos remotos / funções disponibilizados pelo servidor: joga (simula uma atividade/jogada realizada pelo usuário), desiste (simula uma desistência da jogada pelo usuário) e finaliza (encerra o registro do usuário).

As interfaces foram disponibilizadas pelo Prof. Sergio Johann Filho e utilizadas durante a implementação, segue abaixo:

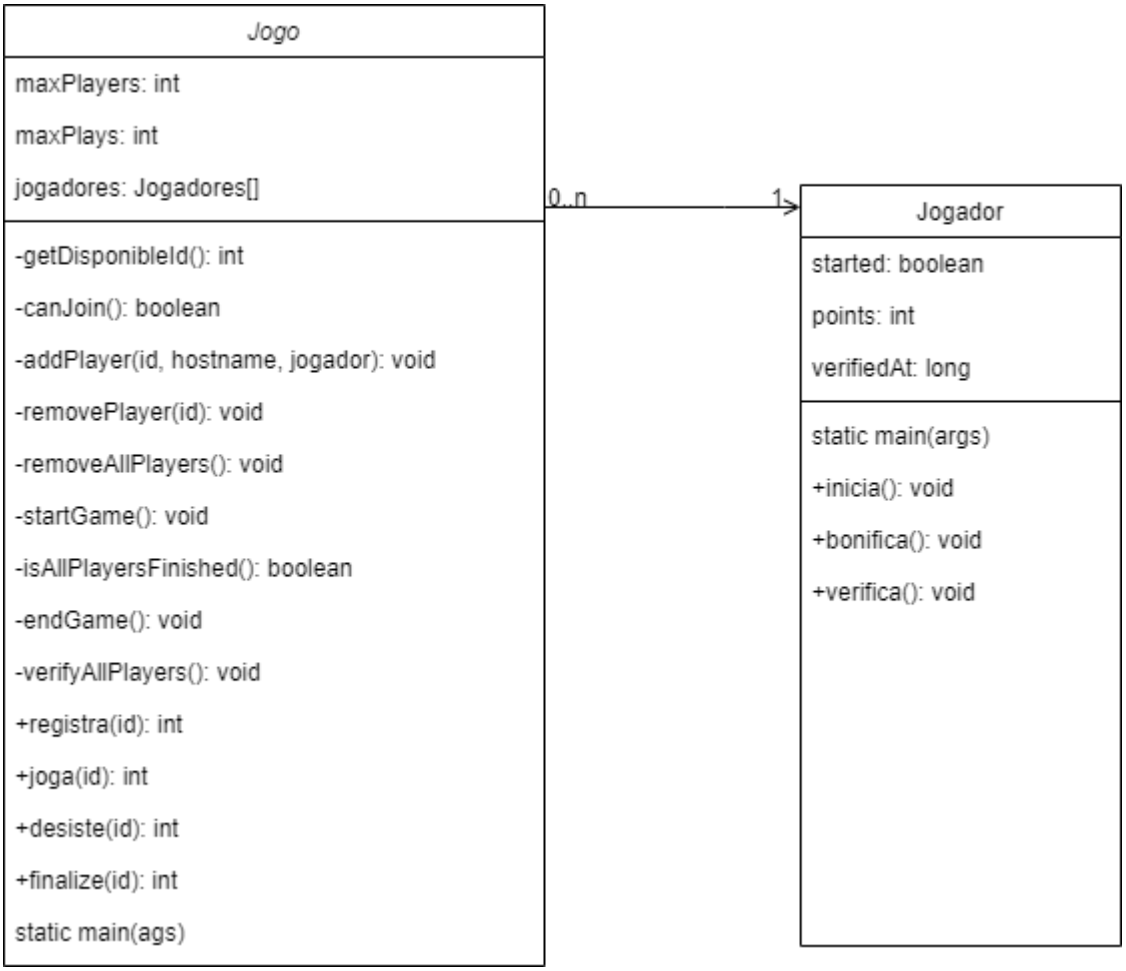
```
public interface JogoInterface extends Remote {
    public int registra() throws RemoteException;
    public int joga(int id) throws RemoteException;
    public int desiste(int id) throws RemoteException;
    public int finaliza(int id) throws RemoteException;
}

public interface JogadorInterface extends Remote {
    public void inicia() throws RemoteException;
    public void bonifica() throws RemoteException;
    public void verifica() throws RemoteException;
}
```

O servidor deve ser iniciado com um parâmetro, definindo o número de N jogadores. Quando todos os jogadores completarem o registro, o servidor deverá invocar o método inicia() de cada jogador.

Cada jogador deve realizar M jogadas (por exemplo 50, invocação do método joga()) e após finalizar (invocação do método finaliza()). Cada jogada realizada pelos jogadores deve ocorrer com um intervalo entre 250ms e 950ms, sendo esse tempo gerado aleatoriamente. A cada jogada, existe uma probabilidade de 3% do servidor dar um bônus ao jogador (invocação do método bonifica(), disponibilizado na interface do jogador). O servidor deve verificar cada jogador a cada 5 segundos, para saber se o mesmo continua ativo.

3. Implementação

Basicamente foi desenvolvido duas classes (“Jogo” e “Jogador), podemos ver seus métodos e atributos principais pelo diagramas de classe abaixo:



A classe “Jogo” é o servidor na topologia, controla quando deve-se realizar as ações dos jogadores e verifica a conexão entre eles. Há três argumentos de configuração que devem ser passados para o servidor, são eles:

- maxPlayers: Quantidade de jogadores na partida, irá iniciar automaticamente quando o número de jogadores estiverem conectados e registrados;
- maxPlays: Quantidade de jogadas realizadas pelo usuário e ponto de parada da aplicação;

O controle do jogo se encontra no método static main, onde podemos encontrar o loop principal que coordena o status da partida:

```
while (true) {

  // inicia jogo automaticamente quando as vagas estiverem preenchidas
  if (!jogo.canJoin() && !jogo.started) {
    jogo.startGame();
  }       You, seconds ago • Uncommitted changes

  // finaliza o jogo caso todos os jogadores ja tenham finalizado
  if (jogo.started && jogo.isAllPlayersFinished()) {
    jogo.endGame();
  }

  // verifica os jogadores a cada 5 segundos
  if (System.currentTimeMillis() - verifiedAt >= 5000) {
    jogo.verifyAllPlayers();
    verifiedAt = System.currentTimeMillis();
  }

  // sleep para liberar a thread
  Thread.sleep(1);
}
```

Na imagem abaixo podemos visualizar a implementação dos métodos remotos que a classe “Jogador” disponibiliza:

```
@Override
public void inicia() throws RemoteException {
  if (started)
    return;
  started = true;
}

@Override
public void bonifica() throws RemoteException {
  if (!started)
    return;
  points += 1;
}

@Override
public void verifica() throws RemoteException {
  verifiedAt = System.currentTimeMillis();
}
```

A classe “Jogador” é o cliente na topologia, realiza a comunicação com o servidor e também libera sua entidade para que o servidor possa realizar invocações de métodos remotos em cada jogador conectado. Na imagem abaixo podemos visualizar os métodos que o “Jogador” pode invocar remotamente do “Jogo”:

```java
public int registra() throws RemoteException {
  if (!canJoin()) {
    return -1;
  }
  int id = getDisponibleId();
  try {
    String hostname = getClientHost();
    String connectLocation = "rmi://" + hostname + ":" + port + "/jogador";
    System.out.println("Connecting to player at : " + connectLocation);
    JogadorInterface jogador = (JogadorInterface) Naming.lookup(connectLocation);
    addPlayer(id, hostname, jogador);
    System.out.println("Receive register from " + hostname + ", id=" + id);
    return id;
  } catch (Exception e) {
    System.out.println("Failed to register player: " + e);
    return -1;
  }
}

public int joga(int id) throws RemoteException {
  System.out.println("Receive (joga) from player #" + id);
  if (!started || ids[id] == -1 || plays[id] >= maxPlays || finished[id])
    return -1;
  plays[id] += 1;
  System.out.println("Player #" + id + " played " + plays[id] + " times");
  if (Math.random() <= 0.03) {
    jogadores[id].bonifica();
    points[id] += 1;       You, 3 hours ago • first commit
    System.out.println("Player #" + id + " received bonification points=" + points[id]);
  }
  return plays[id];
}

public int desiste(int id) throws RemoteException {
  if (!started || ids[id] == -1 || finished[id]) {
    return -1;
  }
  removePlayer(id);
  return 0;
}

public int finaliza(int id) throws RemoteException {
  if (!started || ids[id] == -1 || plays[id] < maxPlays || finished[id]) {
    return -1;
  }
  finished[id] = true;
  return 0;
}
```

Abaixo seu loop principal de execução, que é responsável por realizar as jogadas enquanto a partida estiver ativa e não ter atingido o limite máximo de jogadas:

```
while (true) {

  // executa jogadas caso a partida tenha iniciado
  if (jogador.started) {

    // trava execução durante 250ms à 950ms
    int randomTime = ThreadLocalRandom.current().nextInt(250, 950);
    Thread.sleep(randomTime);

    int played = jogo.joga(id);
    System.out.println("'Joga' executed id=" + id);

    // caso tenha atingido limite de jogadas, então finaliza a partida
    if (played == -1) {
      jogo.finaliza(id);
      jogador.started = false;
      System.out.println("'Finaliza' executed id=" + id);
    }
  }

  // sleep para liberar thread
  Thread.sleep(1);
}
```

4. Utilização do programa

Deve-se executar primeiro a aplicação do jogo, realizando o comando no terminal da seguinte forma:

```
java Jogo <server ip> <num players> <play amount>
```

Onde o primeiro parâmetro é o ip do servidor, para que libere o acesso a classe remota neste IP, o segundo parâmetro é a quantidade de jogadores que é necessário para iniciar o jogo e o terceiro parâmetro a quantidade de jogadas de cada jogador. Ao executar com sucesso, o output no terminal deve ser semelhante a este:

```
java RMI registry created.
Server is ready at: rmi://192.168.15.4:52369/jogo
```

Agora precisamos iniciar os jogadores, para isso deve realizar o comando no terminal da seguinte forma:

```
java Jogador <server ip> <client ip>
```

Onde o primeiro parâmetro é o ip do servidor e o segundo o ip do próprio cliente, isto é necessário para poder realizar a comunicação bidirecional entre as classes jogador e jogo. Após a execução do comando, o jogador irá conectar no servidor e automaticamente realizar seu registro.

```
Java RMI registry already exists.
Connecting to server at: rmi://192.168.15.4:52369/jogo
Connected to server
Player server is ready on rmi://192.168.15.4:52369/jogador
Registered to server, received id= 0
```

Perceba que pelos logs foi aberto as duas camadas de comunicação entre Jogo e Jogador, e logo após essa conexão ser estabelecida, o cliente realiza seu registro e o servidor retorna um id para que possa identificá-lo em suas futuras ações. Abaixo temos alguns logs do jogo em execução, recebendo as jogadas do jogador e finalizando o jogo ao final, neste exemplo foi utilizado apenas um jogador para maior facilidade de entendimento do relatório.

```
Player #0 played 7 times
Receive (joga) from player #0
Player #0 played 8 times
Receive (joga) from player #0
Player #0 played 9 times
Player #0 received bonification points=6
Receive (joga) from player #0
Player #0 played 10 times
Receive (joga) from player #0

Player #0 have 6 points

##################################################################
###### Finished game, waiting players to start new game #######
##################################################################
```