Pontifícia Universidade Católica do Rio Grande do Sul – PUCRS
Curso de Ciência da Computação
Programação Distribuída – Turma 128 – Prof. Roland Teodorowitsch – 2019/1

TRABALHO DA ÁREA 1: SPECULATE DISTRIBUÍDO EM JAVA RMI

Objetivo

Implementar uma aplicação distribuída em Java RMI (*Remote Method Invocation*) que permita que usuários remotos disputem o jogo Speculate (JOGOS, [s.d.], p. 61).

Regras do Jogo

Speculate é um jogo de dados tradicional baseado em apostas, muito popular em *pubs* e clubes fechados (JOGOS, [s.d.], p. 61).

O jogo consiste de 33 bolas, 1 dado e um tabuleiro (tal como o tabuleiro mostrado na Figura 1). Na parte central do tabuleiro há espaço armazenar bolas e há também casas numeradas de 1 até 5 que podem conter alguma das bolas e uma canaleta numerada com 6, que permite que bolas largadas nela caiam na parte central do tabuleiro (juntamente com as demais bolas).

Figura 1 – Tabuleiro do jogo Speculate
(http://gameanalyticz.blogspot.com.br/2008_10_01_archive.html)

Uma partida de Speculate pode ser disputada por dois ou mais jogadores, mas para simplificar o desenvolvimento será considerado que a partida ocorrerá sempre entre 2 jogadores. Inicialmente 3 bolas são colocadas nas casas 1, 3 e 5 do tabuleiro. As demais 30 bolas são divididas entre os participantes (15 bolas para cada jogador).

O primeiro jogador a se habilitar (registrar) para a partida, inicia jogando. E o objetivo dos jogadores será eliminar as bolas que cada um tem nas mãos. Antes de realizar suas jogadas, o jogador deve dizer quantas vezes quer lançar o dado. O número de lançamentos deve variar de 1 até o número de bolas que este jogador tem em suas mãos. Para cada lançamento de dado, o número obtido indica em que casa a bola deve ser colocada. Se a casa estiver livre, o jogador coloca uma de suas bolas ali. Se a casa estiver ocupada, o jogador deve pegar esta bola e colocá-la junto com as bolas que tem em suas mãos, deixando a respectiva casa livre. Se o jogador tiver obtido o número 6, ele coloca uma de suas bolas na respectiva cavidade e ela rola definitivamente para o centro do tabuleiro.

Depois de executar todos os seus lançamentos, o jogador passa a vez para o adversário, que procede exatamente da mesma forma (definindo o número de lançamentos e executando estes lançamentos).

O vencedor será o primeiro jogador a ficar sem nenhuma bola.

Definição

Sua tarefa é desenvolver uma aplicação distribuída em **Java RMI** formada por dois processos, um processo-cliente e um processo-servidor, que seja capaz de executar partidas do jogo Speculate, tal como descrito no início deste documento. Quando o servidor for iniciado, ele deverá ser criado de forma que se possa disputar até 500 partidas simultâneas entre 2 jogadores.

No funcionamento normal da aplicação, para disputar uma partida, cada jogador deverá registrar-se (informando o nome e recebendo um identificador que será usado nas demais operações remotas). O primeiro jogador registrado deverá, naturalmente, esperar que um segundo jogador se registre para que a sua respectiva partida seja iniciada. Assim que o segundo jogador se registrar, a partida inicia com turnos alternados (definição do número de jogadas e jogadas). Enquanto isso o servidor continua recebendo o registro de novos jogadores e organizando-os em novas partidas, conforme a ordem de registro.

Uma partida encerra quando um dos jogadores conseguir livrar-se de todas as suas bolas (conforme as regras do jogo).

Nesta aplicação distribuída, o processo-servidor deverá funcionar de modo que:

- sejam suportadas 500 partidas simultâneas de Speculate entre 2 jogadores devidamente registrados (ou identificados) no servidor;
- quando um jogador se registra, ele deverá esperar que outro jogador também se registre (quando o próximo jogador se registrar, será formada uma dupla que disputará a próxima partida);
- o primeiro jogador a se registrar inicia jogando;
- responda a invocações remotas de métodos realizadas pelos clientes (por exemplo, conforme a descrição de operações descrita a seguir);
- haja limites de tempo para determinados eventos: 2 minutos (120 segundos) pelo registro do segundo jogador; 60 segundos pelas jogadas de cada jogador; e 60 segundos para “destruir” a partida depois de definido o vencedor.

O cliente será responsável: pela interface com o usuário (que poderá ser tanto em modo texto quanto em modo gráfico); e por executar as invocações remotas de métodos disponíveis no servidor, de modo que os usuários possam jogar partidas consistentes.

Quanto à estrutura da aplicação, deve-se usar os conceitos de orientação a objetos e desenvolvimento de aplicações usando RMI (*Remote Method Invocation*). Na programação usando RMI, o servidor deverá instanciar um objeto (por exemplo, SpeculateImpl) que terá seus métodos invocados remotamente. Os métodos que se deseja invocar remotamente no servidor devem ser declarados em uma interface. A classe do objeto instanciado pelo servidor (que terá seus métodos invocados remotamente) corresponde

à implementação desta interface.

Como tipicamente é feito nas aplicações que usam Java RMI, o processo-servidor deverá invocar `Naming.rebind`, instanciando um objeto e registrando-o junto ao processo que controla a comunicação via RMI (*rmiregistry*). A partir disto, o servidor estará em funcionamento e o objeto registrado estará apto a receber invocações remotas dos métodos declarados na sua interface.

O processo-cliente, por sua, vez usará `Naming.lookup` para criar uma referência ao objeto remoto, o que permitirá que o cliente acesse os métodos do objeto que está no servidor.

Serão necessárias, no mínimo, classes para:
- processo-servidor (chamada, por exemplo, de `SpeculateServer`, esta classe cria a instância de um objeto que terá seus métodos chamados remotamente),
- processo-cliente (chamada, por exemplo, de `SpeculateClient`, esta classe é responsável pela interação entre usuário e processo-servidor),
- definição da interface (chamada de `SpeculateInterface`, declara os métodos que serão invocados remotamente),
- definição de uma classe cujo(s) objeto(s) terá(ão) seus métodos invocados remotamente (provavelmente chamada de `SpeculateImpl`, esta classe conterá a implementação da interface, bem como estruturas de dados e outros métodos locais necessários para a resolução do problema).

É muito provável que o desenvolvedor chegue à conclusão que é necessário contar também com outras classes auxiliares, como, por exemplo, uma classe para o tabuleiro, uma classe para dados, etc. Detalhes como este são específicos de cada implementação.

Operações

As seguintes operações remotas deverão ser implementadas pelo servidor:

1) **registraJogador**
   Recebe: *string* com o nome do usuário/jogador.
   Retorna: id (valor inteiro) do usuário (que corresponde a um número de identificação único para este usuário durante uma partida), -1 se este usuário já está cadastrado ou -2 se o número máximo de jogadores (2 vezes o número máximo de partidas) tiver sido atingido.

2) **encerraPartida**
   Recebe: id do usuário (obtido através da chamada **registraJogador**).
   Retorna: código de sucesso (0 indica sucesso e -1, erro).
   Observação: caso um dos jogadores chame **encerraPartida** antes de se determinar um vencedor para a partida ~~ou de se determinar que houve empate~~, o outro jogador será vencedor por WO (ou seja, receberá o código 5 quando chamar **ehMinhaVez**).

3) **temPartida**

Recebe: id do usuário (obtido através da chamada **registraJogador**).
Retorna: -2 (tempo de espera esgotado), -1 (erro), 0 (ainda não há partida), 1 (sim, há partida e o jogador inicia jogando) ou 2 (sim, há partida e o jogador é o segundo a jogar).

4) **obtemOponente**
Recebe: id do usuário (obtido através da chamada **registraJogador**).
Retorna: *string* vazio para erro ou *string* com o nome do oponente.

5) **ehMinhaVez**
Recebe: id do usuário (obtido através da chamada **registraJogador**).
Retorna: -2 (erro: ainda não há 2 jogadores registrados na partida), -1 (erro: jogador não encontrado), 0 (não), 1 (sim), 2 (é o vencedor), 3 (é o perdedor), ~~4 (houve empate),~~ 5 (vencedor por WO), 6 (perdedor por WO).

6) **obtemNumBolas**
Recebe: id do usuário (obtido através da chamada **registraJogador**).
Retorna: número de bolas que o jogador ainda tem em suas mãos, -2 (erro: ainda não há 2 jogadores registrados na partida), -1 (erro: jogador não encontrado).

7) **obtemNumBolasOponente**
Recebe: id do usuário (obtido através da chamada **registraJogador**).
Retorna: número de bolas que o oponente ainda tem em suas mãos, -2 (erro: ainda não há 2 jogadores registrados na partida), -1 (erro: jogador não encontrado).

8) **obtemTabuleiro**
Recebe: id do usuário (obtido através da chamada **registraJogador**).
Retorna: *string* vazio em caso de erro ou *string* com o tabuleiro de jogo.
Observação: essa *string* é uma representação do tabuleiro que possui 6 caracteres, respectivamente correspondentes ao estado de cada uma das 6 casas do tabuleiro. Se o caractere corresponder a um "*", isto significa que a respectiva casa está ocupada por uma bola. Se a casa estiver desocupada, o caractere será o próprio valor do dado que deve ser tirado para colocar uma bola nesta casa. A casa 6, por exemplo, nunca conterá um "*". Como no início do jogo há bolas nas casas 1, 3 e 5, o *string* retornado no início do jogo deverá ser "*2*4*6".

9) **defineJogadas**
Recebe: id do usuário (obtido através da chamada **registraJogador**), número de lançamentos que o jogador realizará.
Retorna: 1 (tudo certo), ou -1 (erro), -2 (erro: ainda não há partida), -3 (não é a vez do jogador), -4 (é a vez do jogador, mas não para definir o número de lançamentos), -5 (o número de jogadas é inválido, por exemplo, maior do que o número de bolas que o jogador tem em mãos).

10) **jogaDado**

Recebe: id do usuário (obtido através da chamada **registraJogador**).

Retorna: número obtido no dado, ou -1 (erro), -2 (erro: ainda não há partida), -3 (não é a vez do jogador), -4 (é a vez do jogador, mas não para jogar dados).

Naturalmente, poderão ser implementadas outras operações ou variações das operações sugeridas.

Avaliação

Trabalhos com trechos copiados integralmente ou parcialmente serão avaliados com a nota mínima (ZERO). Os demais trabalhos serão avaliados numa escala de 0 (ZERO) até 10 (DEZ), levando em consideração as características descritas neste documento. Serão utilizados os seguintes pesos nesta avaliação:

- 40%: a aplicação executou corretamente sem erros, apresentando o comportamento esperado, conforme as regras do jogo.
- 20%: todas as particularidades (tais como número de partidas e usuários, etc.) definidas neste documento foram implementadas.
- 20%: o aluno usou padrões de programação adequados (programação estruturada, nomes de variáveis significativos, comentários, etc.).
- 10%: usou bloqueios para proteger variáveis compartilhadas contra inconsistências referentes a acesso concorrente;
- 10%: foi implementado um mecanismo de temporização que funciona corretamente?

Entrega

O trabalho deve ser desenvolvido individualmente.

A data de entrega do trabalho é **2 de maio de 2019**.

Cada aluno deverá entregar todos os arquivos com extensão ".java" necessários para compilar e executar o projeto. E também deverá apresentar a execução de sua aplicação para o professor.

Em caso de cópia de trabalhos serão avaliados com a nota mínima (zero).

REFERÊNCIAS

JOGOS de Todo Mundo. [Florianópolis]: SESC Santa Catarina, [s.d.]. 64 p.