Prof.ª Ana Cristina Barreiras Kochem Vendramin
Universidade Tecnológica Federal do Paraná (UTFPR), Câmpus Curitiba
Engenharia de Computação, Sistemas de Informação
Programa de Pós-Graduação em Computação Aplicada (PPGCA)
Departamento Acadêmico de Informática (DAINF)

Chamada bidirecional de métodos em **Java RMI**

Passos para desenvolver uma aplicação com chamada bidirecional de métodos em **Java RMI**. Essa aplicação servirá como base para o desenvolvimento da aplicação que envolve eventos e notificações.

O pacote **java.rmi.registry** tem uma classe chamada **LocateRegistry** (http://docs.oracle.com/javase/8/docs/api/java/rmi/registry/LocateRegistry.html) e uma interface **Registry** (http://docs.oracle.com/javase/8/docs/api/java/rmi/registry/Registry.html). Ambas servem para acessar o serviço de nomes do Java RMI.

Existem 3 formas de iniciar o serviço de nomes do Java RMI:

1. chamando o executável **rmiregistry** via linha de comando;
2. chamando **rmiregistry** dentro do código através do método exec() da classe Runtime; ou
3. usando a classe **LocateRegistry**.

Para utilizar as opções (1) e (2) é necessário chamar **Naming**.metodo() como nos slides do arquivo 9_JavaRMI.pdf, sendo que os métodos disponíveis no serviço de nomes são: bind(), rebind(), unbind(), lookup() e list(). (https://docs.oracle.com/javase/7/docs/api/java/rmi/Naming.html)

Peço que utilizem a opção (3) e sigam os seguintes passos:

1. Criação de 2 **projetos** distintos: um para o cliente chamado **Cliente_HelloWorld** e outro para o servidor chamado **Servidor_HelloWorld**;

2. Criação de 2 **pacotes** com o mesmo nome, um em cada projeto criado no passo 1: um pacote para o cliente chamado **HelloWorld** e outro pacote para o servidor chamado **HelloWorld**;

3. Criação da **interface do processo Servidor**, chamada **InterfaceServ**. Essa interface deve estender a interface Remote e conter a assinatura de um método chamado registrarInteresse(String texto, InterfaceCli

referenciaCliente). Esse método possui dois parâmetros de entrada: uma String qualquer e a referência do cliente (cujo tipo é a interface remota do cliente chamada InterfaceCli).

ATENÇÃO: a interface do servidor deve estar no pacote HelloWorld do servidor e do cliente. Esse é o desafio de abertura, isto é, a divulgação da interface remota para que os processos remotos possam invocar os métodos definidos nessa interface.

4. Criação da **interface do processo Cliente**, chamada **InterfaceCli**. Essa interface deve estender a interface Remote e conter a assinatura de um método chamado notificar(String texto). Esse método possui um único parâmetro de entrada: uma String qualquer;

   ATENÇÃO: a interface do cliente deve estar no pacote HelloWorld do cliente e do servidor. Esse é o desafio de abertura, isto é, a divulgação da interface remota para que os processos remotos possam invocar os métodos definidos nessa interface.

5. Criação da **classe servente do servidor**, chamada **ServImpl**. Essa classe precisa estender a classe **UnicastRemoteObject** e implementar a interface InterfaceServ. Quando um cliente invocar o método registrarInteresse(), o servidor receberá uma string qualquer e a referência do cliente. Ao receber esses dois parâmetros, de posse da referência do cliente o servidor invocará o método do cliente notificar() enviando para ele uma string qualquer;

6. Criar a **classe servente do cliente**, chamada **CliImpl**. Essa classe precisa estender a classe **UnicastRemoteObject** e implementar a interface InterfaceCli. Quando o servidor invocar o método notificar(), o cliente apenas mostrará a string recebida na tela.

   O construtor dessa classe receberá a referência do servidor remoto (referenciaServidor, ver passos 8(b) e 8(c)) e com esta referência ele poderá chamar o método registrarInteresse() do servidor e passará uma string qualquer e sua referência (estando na classe CliImpl, basta passar **this** como referência);

   referenciaServidor.registrarInteresse(“Oi”, this);

7. Criar a **classe Servidor** com o método main que:
   a. iniciará o serviço de nomes (utilizando preferencialmente a classe LocateRegistry e o método createRegistry);

Registry referenciaServicoNomes = LocateRegistry.createRegistry(int portaSN);

Após criar o Serviço de Nomes (SN) na máquina local e na porta portaSN (lembrando que a porta default do SN é a 1099), o servidor armazena a referência desse SN na variável referenciaServicoNomes cujo tipo é Registry (isto é, a interface remota do SN). De posse da referência do SN, pode-se

chamar seus métodos bind(), rebind(), unbind(), lookup() e list(). Os métodos utilizados pelo servidor são bind(), rebind() e unbind().

b. instanciará a classe ServImpl. Lembrando que a referência do servidor é o resultado da criação da instância de ServImpl;

InterfaceServ referenciaServidor = new ServImpl();

c. registrará a referência remota da sua aplicação (tipo InterfaceServ) no serviço de nomes.

referenciaServicoNomes.rebind(“HelloWorld”, referenciaServidor);

8. Criar a **classe Cliente** com o método main que:

a. obterá a referência do serviço de nomes (usar preferencialmente a classe LocateRegistry e método getRegistry);

Registry referenciaServicoNomes = LocateRegistry.getRegistry(String maquinaServidor, int portaSN);
ou
Registry referenciaServicoNomes = LocateRegistry.getRegistry(); // assume localhost e porta default 1099.

b. Obterá a referência do servidor, consultando o SN:

InterfaceServ referenciaServidor = (InterfaceServ) referenciaServicoNomes.lookup(“HelloWorld”);

c. instanciará a classe CliImpl passando a referência do servidor como parâmetro.

new CliImpl(referenciaServidor);

Não esquecer:

- Interfaces remotas precisam estender a interface Remote;
- Métodos remotos (na interface e na classe servente) e o construtor das classes serventes devem lidar com a exceção RemoteException.