ATIVIDADE PRÁTICA – JAVA RMI

- Trabalho em dupla ou individual
- Códigos iguais acarreta nota zero para ambas as equipes
- Pode ser utilizado qualquer linguagem de programação tal como: python, c, c++, javascript.

Descrição

Essa atividade mostrará como utilizar o Java RMI para efetuar a comunicação entre objetos distribuídos.

Roteiro

Desenvolvendo um Servidor RMI

Para aceitar chamadas remotas de métodos via RMI, um servidor deve estender a interface java.rmi.Remote e declarar os métodos que serão acessados remotamente.

Na interface RMI abaixo é definido um método hello(), que retorna uma saudação quando é chamado.

```
/** HelloWorld.java **/
import java.rmi.*;

public interface HelloWorld extends Remote {
   public String hello() throws RemoteException;
}
```

A exceção java.rmi.RemoteException indica erros na chamada remota, e deve ser prevista pelos métodos de interfaces RMI.

Agora vamos implementar a interface. Para facilitar o nosso trabalho, vamos usar como base a classe UnicastRemoteObject, que já implementa alguns métodos necessários para o servidor. Temos que criar também um construtor para o nosso servidor (neste caso, ele apenas chama o construtor da classe base).

```
/** HelloServer.java **/

import java.rmi.*;
import java.rmi.server.*;
import java.rmi.registry.*;

public class HelloServer implements HelloWorld {
   public HelloServer() {}
   // main()
   // hello()
}
```

Na função main() do servidor iremos criar um objeto que implementa a interface HelloWorld e registrá-lo como um servidor no registro do RMI, com o nome "HelloWorld", para que ele possa ser localizado pelos clientes.

```
   public static void main(String[] args) {
      try {
         // Instancia o objeto servidor e a sua stub
         HelloServer server = new HelloServer();
         HelloWorld stub = (HelloWorld)
UnicastRemoteObject.exportObject(server, 0);
         // Registra a stub no RMI Registry para que ela seja obtida
pelos clientes
         Registry registry = LocateRegistry.getRegistry();
         registry.bind("Hello", stub);
         System.out.println("Servidor pronto");
      } catch (Exception ex) {
         ex.printStackTrace();
      }
   }
```

Temos que implementar agora os métodos definidos na interface do servidor. Nesta aplicação, há apenas o método hello() para ser implementado.

```
public String hello() throws RemoteException {
   System.out.println("Executando hello()");
   return "Hello!!!";
}
```

Criando um Cliente RMI

Crie um cliente que obtenha uma referência para o servidor no registro RMI e chame o método hello(). Abaixo temos um exemplo de aplicação com interface de texto.

```
/** HelloClient.java **/
import java.rmi.registry.*;
public class HelloClient {
   public static void main(String[] args) {
      String host = (args.length < 1) ? null : args[0];
      try {
         // Obtém uma referência para o registro do RMI
         Registry registry = LocateRegistry.getRegistry(host);

         // Obtém a stub do servidor
         HelloWorld stub= (HelloWorld) registry.lookup("Hello");

         // Chama o método do servidor e imprime a mensagem
         String msg = stub.hello();
         System.out.println("Mensagem do Servidor: " + msg);
      } catch (Exception ex) {
         ex.printStackTrace();
      }
   }
}
```

Compilando e Executando a Aplicação RMI

Edite e compile os arquivos acima usando o compilador Java:

```
> javac *.java
```

No **Windows**, você deve abrir um Prompt de comando, ir até o diretório no qual estão as classes compiladas e iniciar o registro RMI com o comando:

```
> start rmiregistry
```

Em seguida, execute o servidor:

```
> start java HelloServer
```

Finalmente, inicie o cliente com o comando:

```
> java HelloClient localhost
```

Tente também, com a ajuda de um colega, fazer a comunicação entre máquinas diferentes. Para isto, substitua 'localhost' pelo endereço IP da máquina do colega ao chamar o cliente.

No **Linux**, abra um terminal, vá até o diretório no qual estão as classes compiladas e execute o seguinte comando:

```
> rmiregistry &
```

Em seguida, execute o servidor:

```
> java HelloServer
```

Finalmente, abra outro terminal, vá até o diretório no qual estão as classes compiladas e inicie o cliente com o comando:

```
> java HelloClient localhost
```

Prática 1

Você deve criar um servidor que cria e mantém contadores que podem ser acessados remotamente.

O contador deve implementar a interface abaixo:

```
import java.rmi.*;

public interface Counter extends Remote {

   // Retorna o valor atual do contador
   public int getValue() throws RemoteException;

   // Incrementa o contador e retorna seu novo valor
   public int nextValue() throws RemoteException;

}
```

Já o servidor deve implementar a interface:

```
import java.rmi.*;

public interface CounterServer extends Remote {

   // Cria um contador com o valor inicial especificado com parâmetro.
   // Retorna uma referencia com a qual o contador pode ser acessado
```

```
remotamente.
   public Counter createCounter(int initValue) throws RemoteException;

}
```

O método createCounter() deve instanciar um contador e exportá-lo, ou seja, torná-lo acessível remotamente, o que é feito chamando:

```
   UnicastRemoteObject.exportObject(counter,0);
```

... onde counter é uma referência para o contador instanciado, que deve ser retornado pelo método.

Implemente o contador e o servidor de contadores. Desenvolva também um cliente que crie e utilize contadores mantidos no servidor.

Prática 2

- Fazer um sistema cliente-servidor para correção de questionários utilizando RMI.
- O cliente envia ao servidor, datagramas contendo cada um uma resposta do tipo V ou F ao questionário, no seguinte formato:
  - <número da questão>;<número alternativas>;<respostas>
  - Exemplo:
    - 1;5;VVFFV
    - 2;4;VVVV

O servidor lê a mensagem e calcula o número de acertos e erros devolvendo uma resposta simples:

- <número da questão>;<número acertos>;<número erros>