UNIVERSIDADE FEDERAL RURAL DO SEMIÁRIDO - UFERSA
Departamento de Computação - DC
Graduação em Ciência da Computação
Disciplina: Sistemas Distribuídos
Prof.: Paulo Henrique Lopes Silva

Prática 8

1. Exemplos simples usando RMI.

- Interface Hello.java

```
import java.rmi.Remote;
import java.rmi.RemoteException;

/**
 * Um objeto remoto é uma instância de uma classe que implementa uma interface remota.
 *
 * Uma interface remota estende a interface java.rmi.Remote e declara um conjunto de métodos remotos.
 *
 * Cada método remoto deve declarar java.rmi.RemoteException (ou uma superclasse de RemoteException)
 * em sua cláusula de throws, além de quaisquer exceções específicas da aplicação.*/

public interface Hello extends Remote {

    // método a ser implementado
   String ola() throws RemoteException;
}
```

- Classe ImplementaHello.java

```
public class ImplementaHello implements Hello {

        public String ola() {

                return "Olá mundo RMI !";

        }

}
```

- Classe Servidor.java

```
import java.net.InetAddress;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;

public class Servidor {

        public Servidor() {}

        public static void main(String args[]) {
```

```
		try {

			//criar objeto servidor
			ImplementaHello implHello = new ImplementaHello();

			// cria um objeto stub do servidor
			/* O método estático UnicastRemoteObject.exportObject exporta o objeto remoto
			 * fornecido para receber invocações de método remoto de entrada em uma porta
			 * TCP anônima e retorna o stub para o objeto remoto para passas aos clientes.
			 *
			 * Como resultado da chamada de exportObject, o RMI pode começar a escutar em um
			 * novo server socket ou pode usar um server socket compartilhado para aceitar
			 * chamadas remotas de entrada para o objeto remoto.
			 *
			 * RemoteException em caso de falha*/
			Hello stub = (Hello) UnicastRemoteObject.exportObject(implHello, 0);


			// acionar rmiregistry na porta padrão(1099)
			LocateRegistry.createRegistry( Registry.REGISTRY_PORT );

			// associa o stub do servidor (objeto remoto) no rmiregistry

			/* O método estático LocateRegistry.getRegistry, que não recebe
			 * nenhum argumento, retorna um stub que implementa a interface
			 * remota java.rmi.registry.Registry e envia invocações para o
			 * registro (rmiregistry) no host do servidor na porta de registro padrão de 1099.
			 */

			System.out.println(InetAddress.getLocalHost().getHostAddress());
			Registry registro =
LocateRegistry.getRegistry(InetAddress.getLocalHost().getHostAddress());

			/* O método bind é então chamado no stub do registro para vincular
			 * o stub do objeto remoto ao nome "Hello" no registro.*/

			registro.bind("Hello", stub);

			System.err.println(" >> Servidor pronto:");

		} catch (Exception e) {
			System.err.println("Erro no servidor: " + e.toString());
			e.printStackTrace();
		}
	}

}
```

- Classe Cliente.java

```
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.util.Scanner;
```

```
import rmi.servidor.Hello;

public class Cliente {

	private Cliente() {}

	public static void main(String[] args) {

		Scanner teclado = new Scanner(System.in);

		System.out.println("Informe o nome/endereço do RMIRegistry:");
		String host = teclado.nextLine();

		try {


			/* O cliente primeiro obtém a referência para o registro invocando
			 * o método static LocateRegistry.getRegistry com o nome do host.
			 */
			Registry registro = LocateRegistry.getRegistry(host);

			/*
			 * o cliente chama o método lookup (busca) do método remoto no stub do rmiregistry para
			 * obter a referência do objeto remoto do rmiregistry do servidor.
			 */
			Hello stub = (Hello) registro.lookup("Hello");

			/* o cliente invoca o método ola() no stub do objeto remoto, provocando as seguintes
			 * ações:
			 *
			 * lado do cliente abre uma conexão com o servidor usando as informações de host e porta
			 * no stub do objeto remoto e, em seguida, serializa os dados da chamada.
			 *
			 * lado do servidor aceita a chamada recebida, despacha a chamada para o objeto remoto e
			 * serializa o resultado para o cliente.
			 *
			 * lado do cliente recebe, desserializa e retorna o resultado para quem fez a chamada.
			 * */

			String resposta = stub.ola();

			System.out.println("Mensagem de resposta: " + resposta);

			teclado.close();

		} catch (Exception e) {
			System.err.println("Erro no cliente: " + e.toString());
			e.printStackTrace();
		}
	}
}
```

Exercícios:

1. Crie um projeto no Eclipse, codifique, execute e observe o funcionamento dessa aplicação.

2. Considere o protótipo do método ola():

- void ola() **throws** RemoteException.

  E a sua nova implementação:

```
public class ImplementaHello implements Hello {

        public void ola() {

                System.out.println("Bem vindo!");

        }

}
```

  Essa mensagem será impressa no servidor ou no cliente? Modifique o projeto e faça o teste.

3. Implemente um servidor que fornece operações básicas de uma calculadora, como:

- Somar dois números.
- Subtrair dois números.
- Multiplicar dois números.
- Dividir dois números.

  Implemente um cliente para testar a chamada desses métodos.