→ Java RMI - Remote Method Invocation

No modelo de programação orientada a objectos, vimos que um programa consiste numa colecção de objectos que comunicam entre si através da invocação dos seus métodos. Este modelo pode ser generalizado de tal forma que um objecto possa invocar um método de um objecto pertencente a um processo diferente. Teremos então um sistema de objectos distribuídos.

A invocação de métodos entre objectos que pertençam a processos distintos, quer residam na mesma máquina quer não, designa-se por "invocação remota" (Remote Method Invocation ou RMI). À invocação de um método de um objecto no mesmo processo chamamos "invocação local".
Para um objecto A poder invocar localmente um método, de um objecto B, terá que possuir a referência de B. Da mesma forma, para que um objecto possa invocar um método de um objecto remoto terá que possuir a sua referência.

Num sistema de objectos distribuídos, dois conceitos são fundamentais.

"Remote object reference"

- outros objectos podem invocar os métodos de um objecto remoto se tiverem acesso à sua referência remota (remote object reference).

"Remote interface"

- cada objecto remoto tem que possuir uma interface remota (remote interface) que especifica quais dos seus métodos podem ser invocados remotamente.

1 – Vamos implementar uma aplicação distribuída muito simples em que um processo servidor disporá de um método que devolve a data do sistema. O(s) processo(s) cliente deverão poder invocar esse método remotamente, isto é, de outra máquina virtual que poderá, ou não, residir num computador diferente.

A) Comecemos por criar a interface remota que define os métodos passíveis de serem invocados remotamente.

```
public interface RMIInterface extends java.rmi.Remote
{
  public java.util.Date getDate() throws java.rmi.RemoteException;
 }
```

Notas: - a interface tem de ser subclasse da interface java.rmi.Remote;
- cada método de uma interface remota tem de lançar a excepção java.rmi.RemoteException.

B) De seguida vamos criar uma classe que implementa a interface anterior.

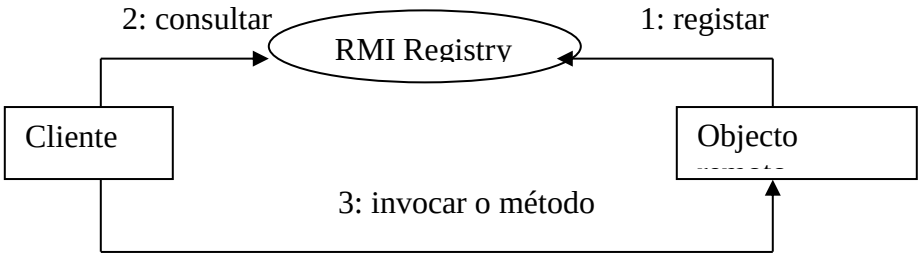
```
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;

public class RMIImpl extends UnicastRemoteObject implements RMIInterface
{
 public RMIImpl(String name) throws RemoteException {
  super();
  try {
   Naming.rebind(name, this);
  }
  catch (Exception e) {
   if (e instanceof RemoteException)
    throw (RemoteException)e;
   else
    throw new RemoteException(e.getMessage());
  }
 }
 public java.util.Date getDate() {
  System.out.println(" Método remoto -- RMIImpl.getDate()");
  return new java.util.Date();
 }
}
```

Notas: - esta classe é subclasse da classe java.rmi.server.UnicastRemoteObject, o que significa que a classe vai ser usada para criar um objeto remoto usando o sistema de comunicação por omissão;
- a classe implementa a interface definida em A;
- o construtor informa o "RMI registry" de que este objecto está disponível com o nome recebido no parâmetro *String name* (Naming.rebind (...).

→ RMI registry

O RMI registry é um serviço de nomes que faz a gestão das referências remotas para o sistema RMI. O servidor tem que registar o objecto no registry, permitindo assim aos clientes obter a referência do objecto remoto.



C) Criar a classe servidor que irá gerar uma instância da classe RMIImpl definida em B).

```
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;

public class RMIServer
{
 public static void main(String[] argv) {
  System.setSecurityManager(new SecurityManager());

  try { //Iniciar a execução do registry no porto desejado

      java.rmi.registry.LocateRegistry.createRegistry(1099);

      System.out.println("RMI registry ready.");

      } catch (Exception e) {
            System.out.println("Exception starting RMI registry:");
            e.printStackTrace();
   }

  try {
   RMIImpl implementaInterface = new RMIImpl("RMIImpl");
   System.out.println("Servidor está OK");
  }
  catch (Exception e) {
   System.out.println("Erro no servidor " + e);
  }
 }
}
```

Notas: - Esta classe contém o método main do servidor;
- Instala um gestor de segurança SecurityManager();
- Inicia a execução do Registry
- Cria uma instância da classe, RMIImpl, que implementa o método remoto;
- Depois de execução do servidor o objecto remoto estará acessível aos clientes pelo nome: "//<máquina onde executa o servidor>/RMIImpl" .

D) Criar a classe cliente

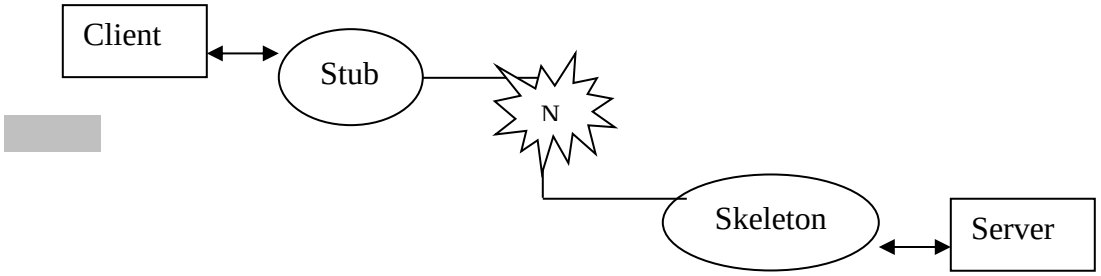
```
import java.rmi.*;
import java.rmi.registry.*;
import java.rmi.server.*;
import java.util.Date;
public class RMIClient
{
  public static void main(String[] argv) {
    String serverName = "";
    System.setSecurityManager(new SecurityManager());
    if (argv.length != 1) {
      try {
        serverName = java.net.InetAddress.getLocalHost().getHostName();
      }
      catch(Exception e) {
        e.printStackTrace();
      }
    }
    else {
      serverName = argv[0];
    }
    if (serverName.equals( "") ) {
      System.out.println("usage: java RMIClient < host running RMI server>");
      System.exit(0);
    }
    try {
      //bind server object to object in client
      RMIInterface  myServerObject =
                    (RMIInterface) Naming.lookup("//"+serverName+"/RMIImpl");

      //invoke method on server object
      Date d = myServerObject.getDate();
      System.out.println("Date on server is " + d);
    }
    catch(Exception e) {
      System.out.println("Exception occured: " + e);
      System.exit(0);
    }
    System.out.println("RMI connection successful");
  }
}
```

Notas: - o processo cliente obtém a referência ao objecto remoto através do método Naming.lookup(), invocando de seguida o método remoto.

→ Stubs and Skeletons

Para podermos estabelecer a ligação entre o cliente e o servidor criados vão ser gerados dois módulos intermédios.



O Stub funciona para o cliente como um "proxy" do objecto remoto. Tem como função tornar transparente a invocação do método remoto. O Stub vai formatar os parâmetros do método de forma a poderem ser transmitidos para o servidor, envia a mensagem e recebe os resultados da execução do método remoto, que depois envia ao cliente. O Skeleton, que existe na máquina do servidor, é responsável por receber os argumentos que vieram do cliente, invocar o método no processo servidor e enviar os resultados.

Para versões da plataforma Java superiores a 5.0 este passo é feito de forma automática.

Pode agora compilar todo o projeto.

E) Permissões de acesso

Para permitir que a sua aplicação rmi possa ser ligada com outras que tentem comunicar com ela, é necessário modificar as permissões existentes por omissão.
Crie um ficheiro de texto com o conteúdo abaixo, com extensão *.policy*, e coloque-o na directoria do seu projecto:

```
grant {
  // allows anyone to listen on un-privileged ports
  permission java.net.SocketPermission "*:1024-65535", "listen,accept,connect";
};
```

F) Parâmetros para a máquina virtual

Finalmente antes de executar o servidor é necessário, além de indicar qual a classe que contém o método main, indicar a localização das classes do projecto, sob a forma de parâmetros para a máquina virtual,

No NetBeans,

Nas **propriedades** do seu projeto, na categoria **Run,**
Em **Working Directory:** introduza: **build/classes**
Em **VM options:**
-Djava.security.policy=nomeDoFicheiro**.policy**

1) Pode agora executar o servidor.

2) Executar o cliente
Se ao executar o servidor obteve a mensagem "Servidor está OK", então o servidor está pronto a executar os pedidos dos clientes.
Para executar o cliente terá que, além de seleccionar a classe que contém o main, indicar a localização do ficheiro .policy.

- Execute várias vezes o processo cliente e verifique que o servidor continua activo.

3) Modifique a aplicação anterior de forma a que o objecto remoto anterior possua um contador que conte o número de invocações do método getDate() num mesmo objeto. Defina um método getCount que devolva o valor desse contador e invoque-o remotamente no cliente.

4) Implemente o exemplo da "Calculadora" estudado na aula teórica T05.

5) Considere agora o exercício 6, da FP02 e redesenhe a aplicação de forma a que um objecto remoto responda às operações implementadas pelo servidor: i) Registar aluno; ii) Consultar quais os alunos registados; iii) Consultar o número de acessos ao servidor …; iv) Dado um nome de aluno devolver o seu número e o seu contacto, …

Para informação adicional consultar:

http://java.sun.com/docs/books/tutorial/rmi/overview.html