→ Java RMI - Remote Method Invocation

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

A invocação de métodos entre objetos 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 objeto no mesmo processo chamamos "invocação local".
Para um objeto A poder invocar localmente um método, de um objeto B, terá que possuir a referência de B. Da mesma forma, para que um objeto possa invocar um método de um objeto remoto terá que possuir a sua referência.

Num sistema de objetos distribuídos, dois conceitos são fundamentais.

"Remote object reference"
- outros objetos podem invocar os métodos de um objeto remoto se tiverem acesso à sua referência remota (remote object reference).

"Remote interface"
- cada objeto 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 contém uma lista de nomes e disponibiliza dois métodos remotos. Um que permite adicionar um nome e outro que permite consultar a lista de nomes. O(s) processo(s) cliente deverão poder invocar esses métodos 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 void adiciona (String s) throws java.rmi.RemoteException;

       public ArrayList<String> consulta() 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 exceção java.rmi.RemoteException.

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

```
import java.rmi.server.UnicastRemoteObject;
import java.util.ArrayList;

public class RMIImpl extends UnicastRemoteObject implements RMIInterface
{
        ArrayList<String> lista;

        public RMIImpl() throws java.rmi.RemoteException {
            super();
            lista = new ArrayList<String>();
        }
        public void adiciona ( String s) throws java.rmi.RemoteException {
            lista.add(s);
        }
        public ArrayList<String> consulta() throws java.rmi.RemoteException{
            return lista;
        }
}
```

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;

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

```
import java.net.*;
import java.rmi.*;
public class RMIServer {
    public static void main(String[] argv) {
        //Se está a trabalhar com uma versão do java inferior a 17,
        //não comente a instrução abaixo:
        //  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:");
          }
```

```
                try {
                        // instanciar objeto remoto
                        RMIInterface objRemoto = new RMIImpl();

                        //registar o objeto remoto no Registry
                        Naming.rebind("RMIImpl", objRemoto);
                        System.out.println("Remote object ready");

                } catch (MalformedURLException | RemoteException e) {

                        System.out.println(e.getMessage());
                }
        }

}
```

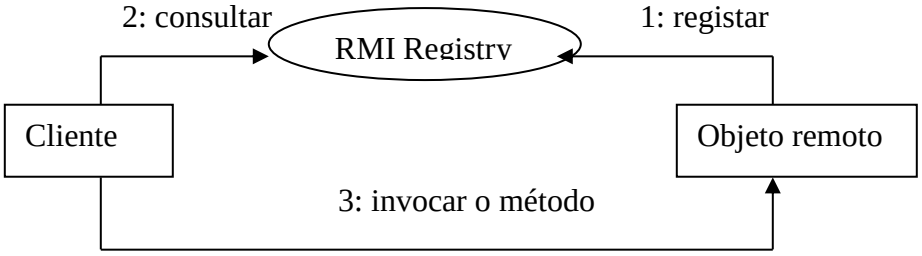
Notas: - Esta classe contém o método main do servidor;

- // *Instala um gestor de segurança SecurityManager();*
- Inicia a execução do Registry;
- Cria uma instância da classe, RMIImpl, que implementa o método remoto;
- - Informa o "RMI registry" de que este objeto está disponível com o nome recebido RMIImpl;

Depois de execução do servidor o objeto remoto estará acessível aos clientes pelo nome: "//<máquina onde executa o servidor>/RMIImpl"

→ 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 objeto no registry, permitindo assim aos clientes obter a referência do objeto remoto.



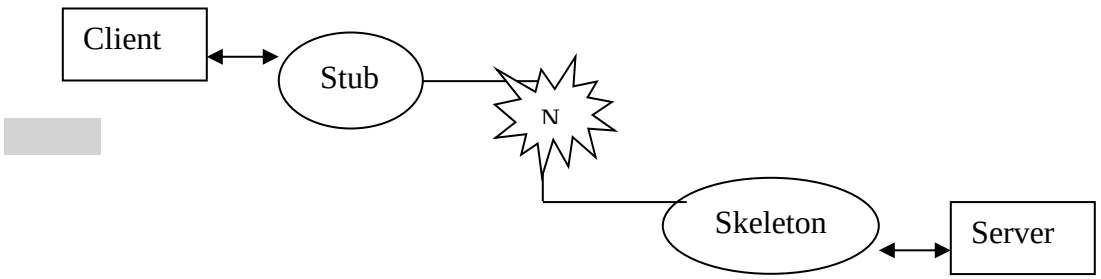
D) Criar a classe cliente

```
import java.rmi.Naming;
public class RMIClient {
   public static void main(String[] argv) {
        //Se está a trabalhar com uma versão do java inferior a 17,
       //não comente a instrução abaixo:
       //System.setSecurityManager(new SecurityManager());

       try {
        //bind server object to object in client
        RMIInterface  myServerObject = (RMIInterface) Naming.lookup("RMIImpl");

        //invoke method on server object
        myServerObject.adiciona("XPTO");

         System.out.println(    myServerObject.consulta()  );
       }
       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 os métodos 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.

```
//Se está a trabalhar com uma versão do java inferior a 17, vá ao final da ficha ver as secções E e F. Depois regresse a este ponto da ficha.
```

1) Pode agora executar o Servidor em "Run as Java application".

Se ao executar o servidor obteve a mensagem "Servidor está OK", então o servidor está pronto a executar os pedidos dos clientes.

2) Executar o cliente.

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

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

4) Modifique o cliente de forma a introduzir um "menu" onde o utilizador pode escolher as opções a executar. As várias opções podem ser selecionadas várias vezes havendo uma opção para terminar o cliente.

5) Modifique a opção inserir, para que seja o utilizador a adicionar o texto que pretende.

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

6) Considere agora o exercício 5 da FP02 e redesenhe a aplicação de forma a que um objeto 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, …

Versões abaixo de Java 17:

E) Permissões de acesso

Para permitir que o cliente possa comunicar com o servidor, é necessário modificar as permissões existentes por omissão.
Crie um ficheiro de texto com o conteúdo abaixo, com extensão *.policy*, e coloque-o na diretoria do seu projecto (ou na diretoria raiz do seu workspace):

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

F) Parâmetros para a máquina virtual

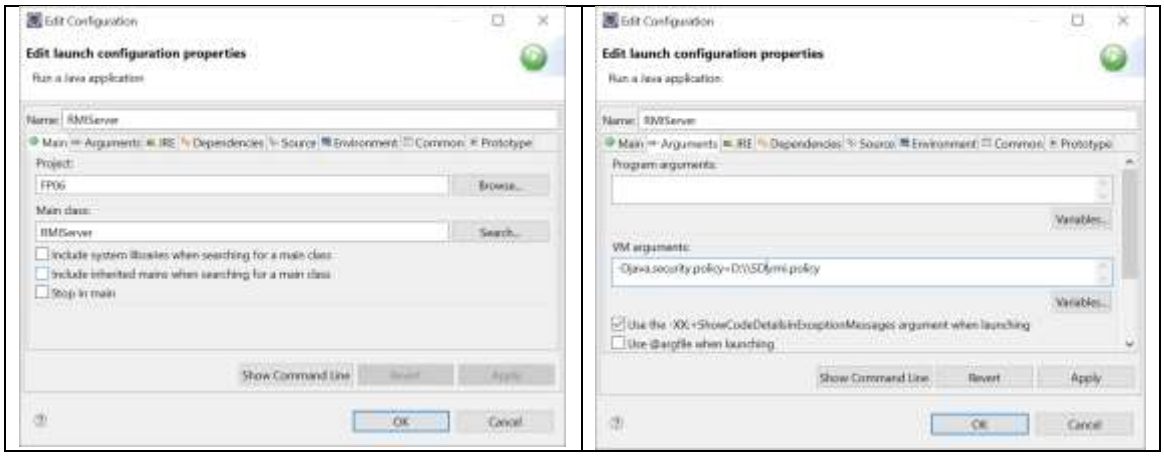
Finalmente antes de executar o servidor é necessário, além de indicar qual a classe que contém o método main, indicar a localização das classes do projecto, sob a forma de parâmetros para a máquina virtual,

No NetBeans,
Nas **propriedades** do seu projeto, na categoria **Run,**
Em **Working Directory:** introduza: **build/classes**
Em **VM options:**
-Djava.security.policy=nomeDoFicheiro**.policy**

No Eclipse (Ver figuras abaixo),

No projeto selecione Properties e em Run/Debug settings, selecionar a classe que contém o main do servidor (RMIServer) em VM arguments inserir:

-Djava.security.policy=nomeDoFicheiro**.policy**



Repetir, para a classe que contém o main do Cliente.