Universidade Federal do Ceará - Campus de Quixadá
Disciplina: Sistemas Distribuídos
Código: QXD0043
Professor: Rafael Braga

Implementação de um serviço remoto simples com RMI

1º Passo: Criação de uma interface remota que proporcionará a comunicação entre cliente e servidor (serviço);

```
import java.rmi.*;
public interface InterfaceRemota extends Remote {
    metodoRemoto1 throws RemoteException;
    ...
    metodoRemotoN throws RemoteException;
}
```

2º Passo: Criação de uma classe Servente que o lado servidor implementará de acordo com a interface remota. As classes Serventes dão "corpo" o serviço fornecido pelo servidor;

```
import java.rmi.*;
public class Servente extends UnicastRemoteObject implements
InterfaceRemota {

    public metodoRemoto1()throws RemoteException{
    //implementacao
    }

    //...
    public metodoRemotoN()throws RemoteException{
    //implementacao
    }
}
```

3º Passo: Criação da classe Servidor (possui um método main()) e publicação do serviço;

```
import java.rmi.*;
public class Servidor{
    public static void main(String args[]){
        System.setSecurityManager(new RMISecurityManager());
        try{
            InterfaceRemota refServente = new Servente();
            Naming.rebind("Apelido_do_Servico", refServente);
            System.out.println("Servidor em execucao");
        }catch(Exception e) {
            System.out.println("ShapeList server main " +
            e.getMessage());
        }
    }
}
```

4º Passo: Criação da classe Cliente que obterá a referência remota para o objeto que implementa o serviço;

```
public class ShapeListClient{
    public static void main(String args[]){
        if(System.getSecurityManager() == null){
            System.setSecurityManager(new RMISecurityManager());
        } else System.out.println("Já há um gerenciador de Seg");
        InterfaceRemota refRemota = null;
        try{
            refRemota =(InterfaceRemota) Naming.lookup("end/apelido");
            System.out.println("Found server");
            refRemota.metodoRemoto1();
            //...
            refRemota.metodoRemotoN();
}
```

5º Passo: Para que o serviço oferecido possa ser acessado remotamente através de RMI, é preciso também as classes auxiliares internas de *stubs* e *skeletons(apenas java 2)*, responsáveis pela comunicação entre o objeto cliente e o objeto servidor. É necessária então, a compilação utilizando o habitual ***javac*** e posteriormente o processo de compilação RMI utilizando o ***rmic*** (para os serventes).

6º Passo: Definição da política de segurança. Java é muito restrito no que diz respeito a comunicação, por questões de segurança, e para conectar uma classe a outra remota é necessário o uso de um arquivo *policy*, que diga ao JVM quais os serviços disponíveis e permitidos àquela classe, como apenas conexão ou fazer download de algum arquivo/classe.

Ex: sem nenhuma restrição

```
grant{
    permission java.security.AllPermission;
};
```

7º **Passo:** Execute o servidor de Nomes (*rmiregistry*), o Servidor e o Cliente. Considere a hierarquia de diretórios do projeto mostrada abaixo para a execução.

```
Projeto
|-src
|   |- Nome_do_pacote
|       |- *.java
|-bin
   |- Nome_do_pacote
       |- *.class, *.stub
```

Considerando que foi criado o pacote: caseRemoto

Compilar os fontes com Javac:
Execute dentro do diretorio que possui os fontes (/src/caseRemoto):

```
javac -d ../../bin/ *.java
```

Criar os stubs com rmic
Execute dentro do diretório que possui os .class (/bin):

```
rmic caseRemoto.Servente
```

Coloque a política no diretório dos .class (/bin)

Rode o serviço de nomes
Execute dentro do diretório que possui os .class (/bin):

```
rmiregistry
```

Obs: Uma alternativa ao comando "rmiregistry" é a inclusão de seguinte instrução no código fonte do servidor:

```
LocateRegistry.createRegistry(1099);
```

Rode o servidor
Execute dentro do diretório que possui os .class (/bin):

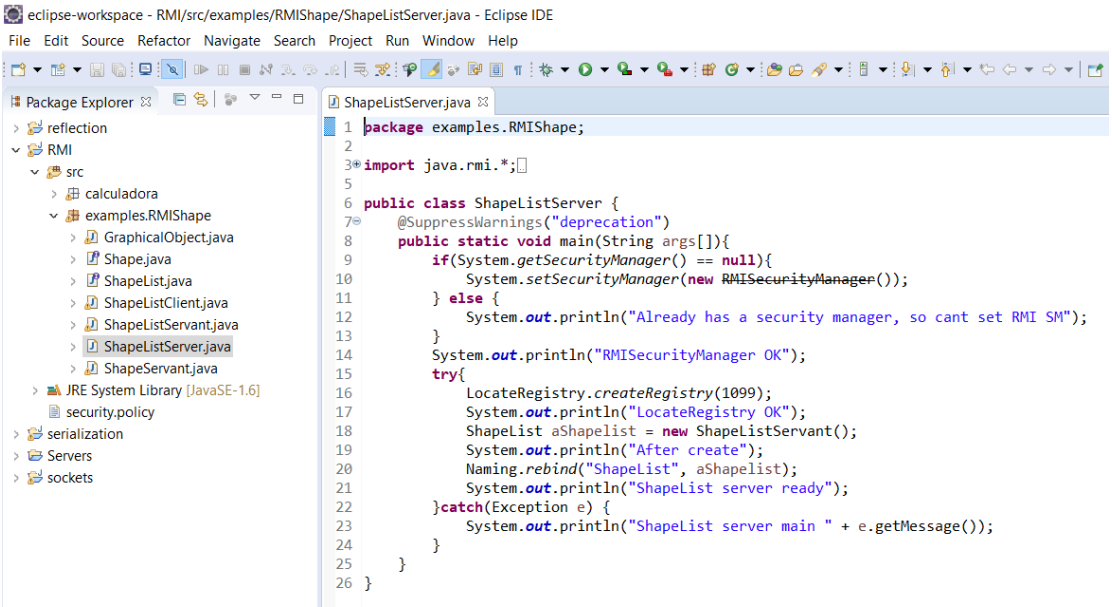
```
java -Djava.server.rmi.codebaseile:///caseRemoto/ -Djava.security.policy=
caseRemoto/policy caseRemoto.Servidor
```

Rode o cliente
Execute dentro do diretorio que possui os .class (/bin):

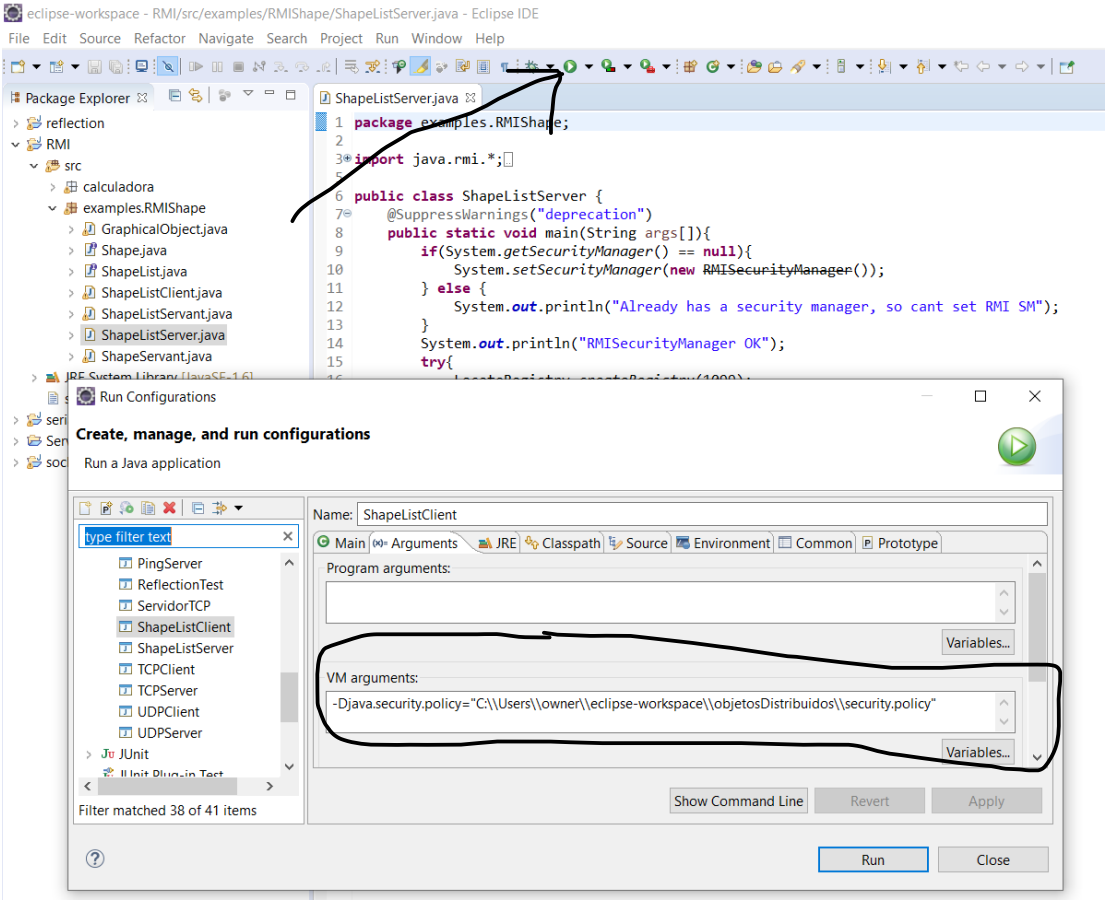
```
java -Djava.security.policy=caseRemoto/policy caseRemoto.Cliente teste
```

Utilizando o Eclipse IDE

Todo processo de criação das classes, compilação (exceto a compilação com o comando rmic para geração do Stub) e execução podem também serem feitos através da IDE Eclipse. Observe a figura abaixo que mostra o exemplo executado em sala, o mesmo do livro, onde todas as classes do exemplo estão criadas.

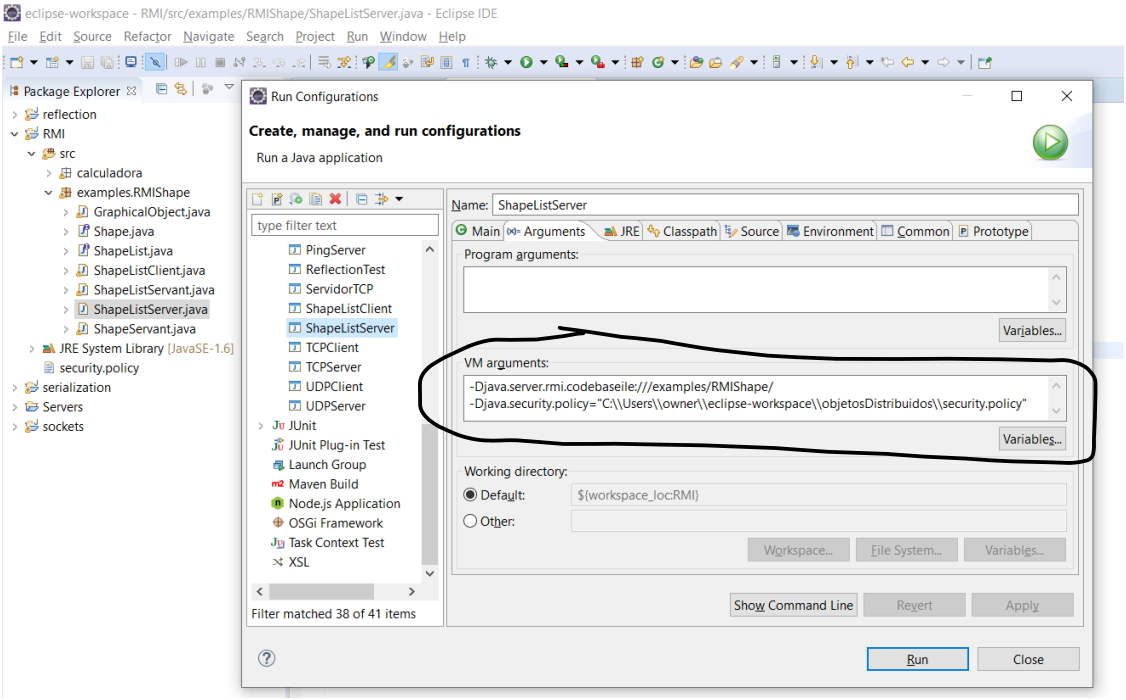


Para a execução do cliente, primeiramente, e necessário informar a localização do arquivo de políticas de segurança, “security.policy”. Para tal, é preciso configurar um profile de execução, no menu de execução (bolinha verde com a seta branca) → “Run Configuration“. Depois dar nome ao profile de execução e informar a localização do arquivo de políticas de segurança na aba “Arguments”. Assim:



No meu caso a localização é: -Djava.security.policy="C:\\Users\\owner\\eclipse-workspace\\objetosDistribuidos\\security.policy".

Para a execução do servidor e necessário informar a localização do arquivo de políticas de segurança, “security.policy” e a localização dos compilados com o arquivo de Stub. Para tal, é preciso configurar um profile de execução, no menu de execução (bolinha verde com a seta branca) → “Run Configuration“. Depois, dar nome ao profile de execução, e na aba “Arguments”, informar a localização do arquivo de políticas de segurança e o caminho dos compilados (.class). Assim:


eclipse-workspace - RMI/src/examples/RMIShape/ShapeListServer.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
reflection
RMI
src
calculadora
examples.RMIShape
GraphicalObject.java
Shape.java
ShapeList.java
ShapeListClient.java
ShapeListServant.java
ShapeListServer.java
ShapeServant.java
JRE System Library [JavaSE-1.6]
security.policy
serialization
Servers
sockets
Run Configurations
Create, manage, and run configurations
Run a Java application
type filter text
PingServer
ReflectionTest
ServidorTCP
ShapeListClient
ShapeListServer
TCPClient
TCPServer
UDPClient
UDPServer
JUnit
JUnit Plug-in Test
Launch Group
Maven Build
Node.js Application
OSGi Framework
Task Context Test
XSL
Filter matched 38 of 41 items
Name: ShapeListServer
Main Arguments JRE Classpath Source Environment Common Prototype
Program arguments:
Variables...
VM arguments:
-Djava.server.rmi.codebaseile:///examples/RMIShape/
-Djava.security.policy="C:\\Users\\owner\\eclipse-workspace\\objetosDistribuidos\\security.policy"
Variables...
Working directory:
Default:
${workspace_loc:RMI}
Other:
Workspace... File System... Variables...
Show Command Line Revert Apply
Run
Close