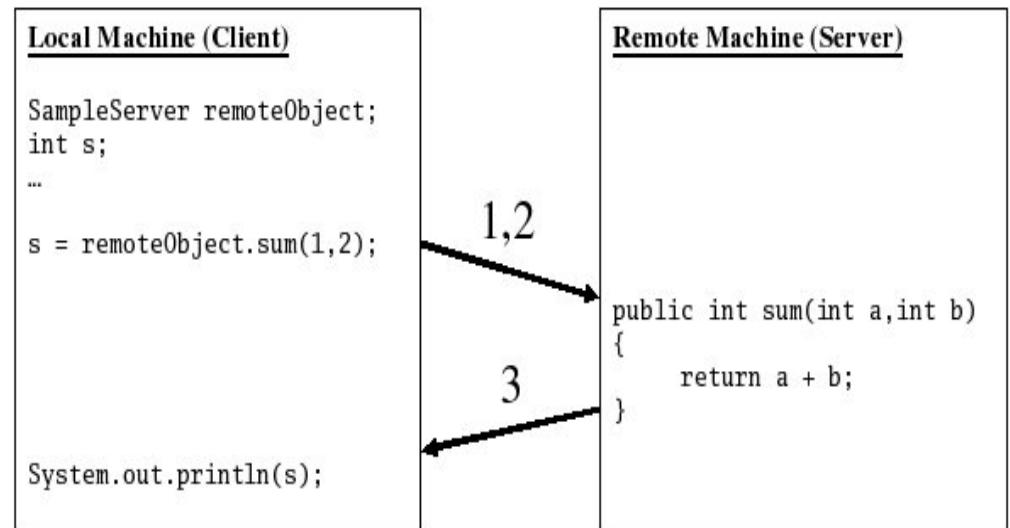




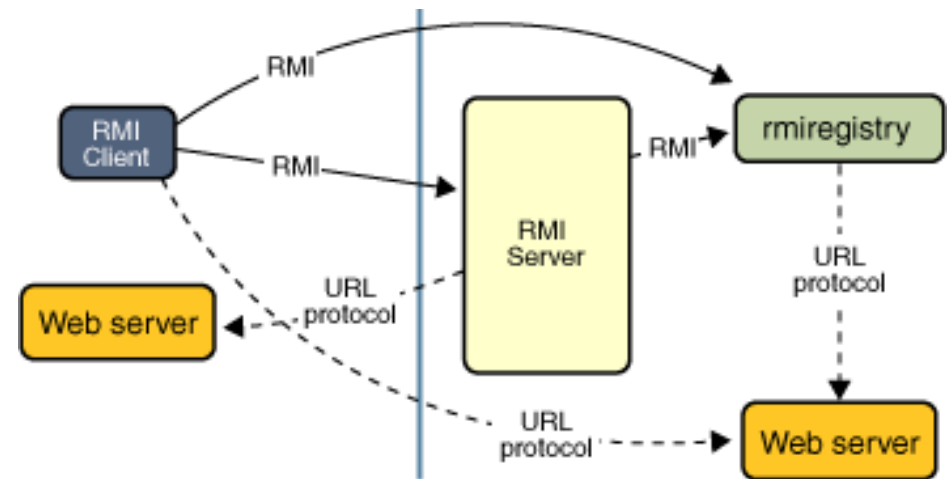
RMI

- Introdução ao RMI
 - RMI é a sigla para Remote Method Invocation.
 - Com RMI é possível utilizar objetos instanciados em uma JVM remota da mesma forma que se estivessemos utilizando um objeto instanciado na JVM local.

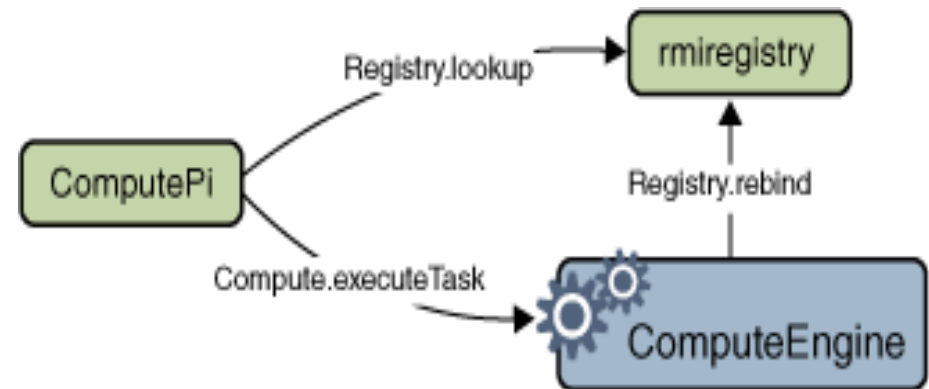


RMI

- Uma aplicação RMI é composta de 2 programas:
 - Um Servidor e
 - Um Cliente
- O Servidor cria os objetos, registra suas referências num servidor e aguarda pela chamada aos métodos dos objetos.
- O Cliente obtém a referência aos objetos remotos e efetua a chamada a seus métodos.



- Construindo um Cliente
 - Um Cliente necessita localizar o objeto remoto antes de poder utilizá-lo.
 - Para localizar um objeto remoto utilizamos o método lookup() de uma das classes abaixo:
 - java.rmi.Naming
 - java.rmi.registry.LocateRegistry
 - java.naming.InitialContext



- `java.rmi.Naming`

- Oferece métodos para armazenar e obter referências à objetos remotos num servidor de registros.

Ex.:

```
Objeto obj = (Objeto)Naming.lookup("rmi://hostname/nome");
```

- hostname é o nome do equipamento remoto onde está o servidor de registro de objetos.
- nome é o nome que faz referência ao objeto que deve ser localizado.

- `java.rmi.registry.LocateRegistry`
 - É utilizado para referenciar um servidor remoto de registro em um equipamento em particular, inclusive o equipamento local.
 - Também é utilizado para criar um servidor de registros de objetos no equipamento local

Ex.:

```
Registry reg = LocateRegistry.getRegistry("hostname");
```

```
Objeto obj = (Objeto)reg.lookup("nome");
```

- `java.naming.InitialContext`
 - Representa o contexto inicial para se efetuar operações de nome (consulta, registro, etc).

Ex:

```
Properties env = new Properties();  
env.put(Context.INITIAL_CONTEXT_FACTORY,  
        "com.sun.jndi.rmi.registry.RegistryContextFactory");  
env.put(Context.PROVIDER_URL, "rmi://hostname:1099");  
Context ctx = new InitialContext(env);  
Objeto obj = (Objeto)ctx.lookup("nome");
```

- Security Manager (java.rmi.SecurityManager)
 - O Security Manager é responsável por gerenciar as atividades da carga dinâmica de objetos e deve ser instalado.

Ex.:

```
System.setSecurityManager(new SecurityManager());
```

Obs.: Quando o SecurityManager é instalado é necessária a configuração das “policies” num dos arquivos:
%JAVA_HOME/lib/security/java.policy ou %USER_HOME/.java.policy.

Ainda há uma outra forma de ativarmos as policies, é a partir da chamada da JVM:

```
java -Djava.security.policy=<nome do arquivo de policy> -jar <nome do jar>
```

Links:

<http://java.sun.com/javase/8/docs/technotes/guides/security/permissions.html>

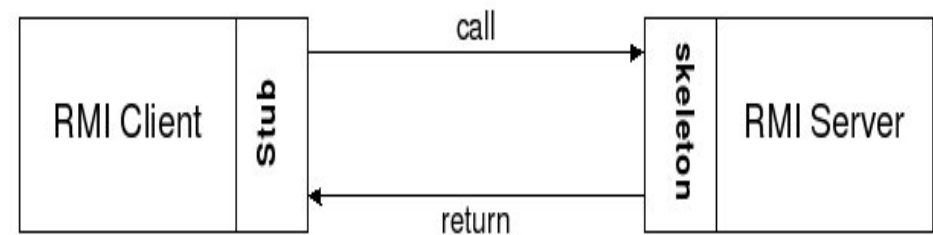
<http://java.sun.com/javase/8/docs/technotes/guides/security/PolicyFiles.html>

- client.policy
 - Este arquivo é utilizado na configuração do security manager e pode ter qualquer nome.
 - Seu conteúdo define as restrições de segurança que são aplicadas à JVM.
 - No caso do cliente RMI temos:

```
grant {  
    permission java.awt.AWTPermission "accessEventQueue";  
    permission java.util.PropertyPermission "java.net.preferIPv4Stack", "write, read";  
    permission java.net.SocketPermission "*:1098-", "connect, resolve";  
};
```

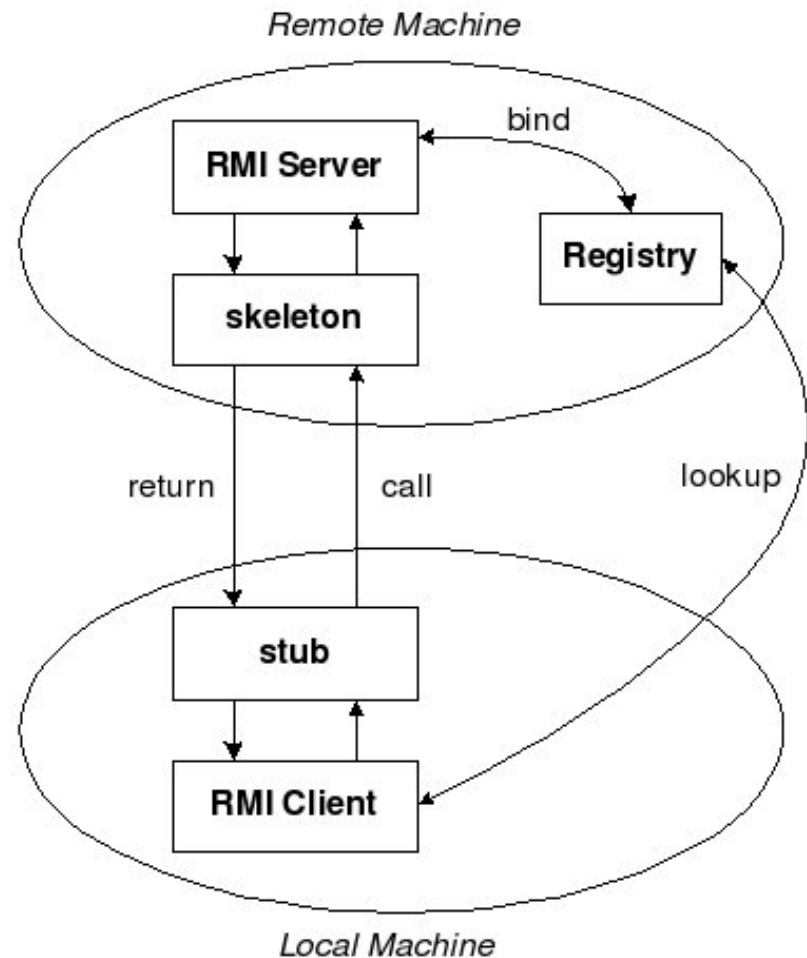
RMI

- Uma vez efetuado o lookup() o próximo passo é efetuar chamadas aos métodos.
- Toda a chamada a métodos remotos podem lançar RemoteException
- Toda a chamada e retorno é efetuada através de stubs e skeleton

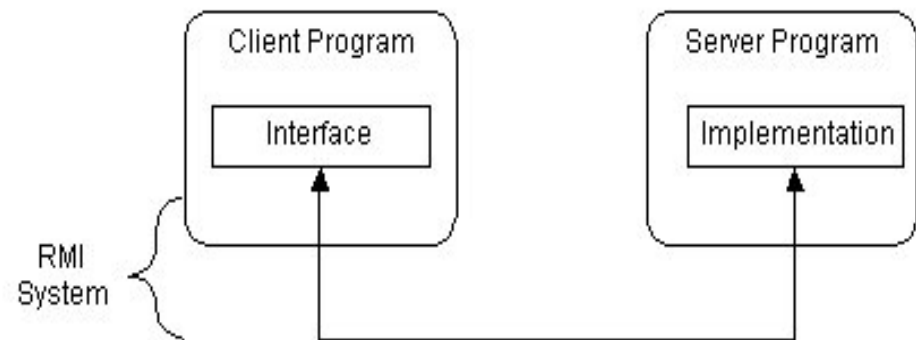


RMI

- O Stub é responsável por criar um Socket conectando ao servidor, serializar os objetos (marshaling) e transmitir ao Skeleton.
- O Skeleton deserializa os objetos recebidos (unmarshaling) e invoca o método do objeto instanciado pelo servidor no momento do registro.



- Construindo um Servidor
 - Um servidor é constituído por duas partes:
 - Uma Interface e
 - Sua implementação
 - A interface tem a responsabilidade de definir os métodos que serão oferecidos aos clientes.
 - A implementação contém a lógica necessária para atender o que foi definido na interface.

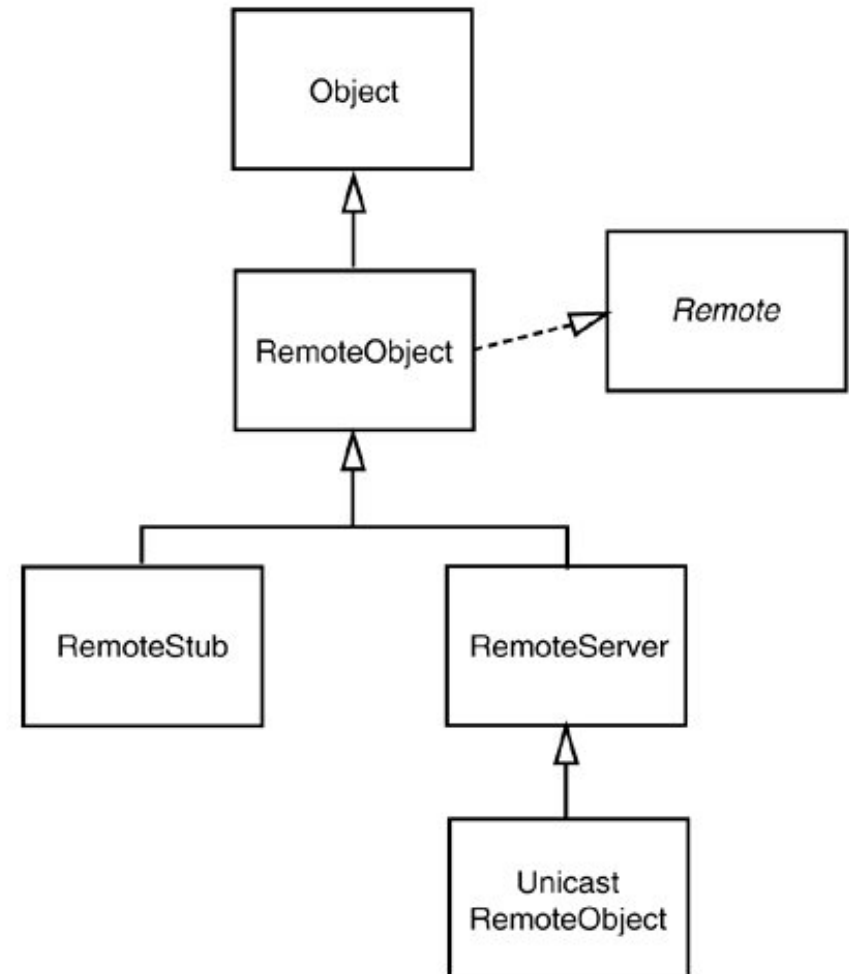


- A Interface

- A interface deve extender:
`java.rmi.Remote`
- Seus métodos devem lançar a exceção:
`java.rmi.RemoteException`

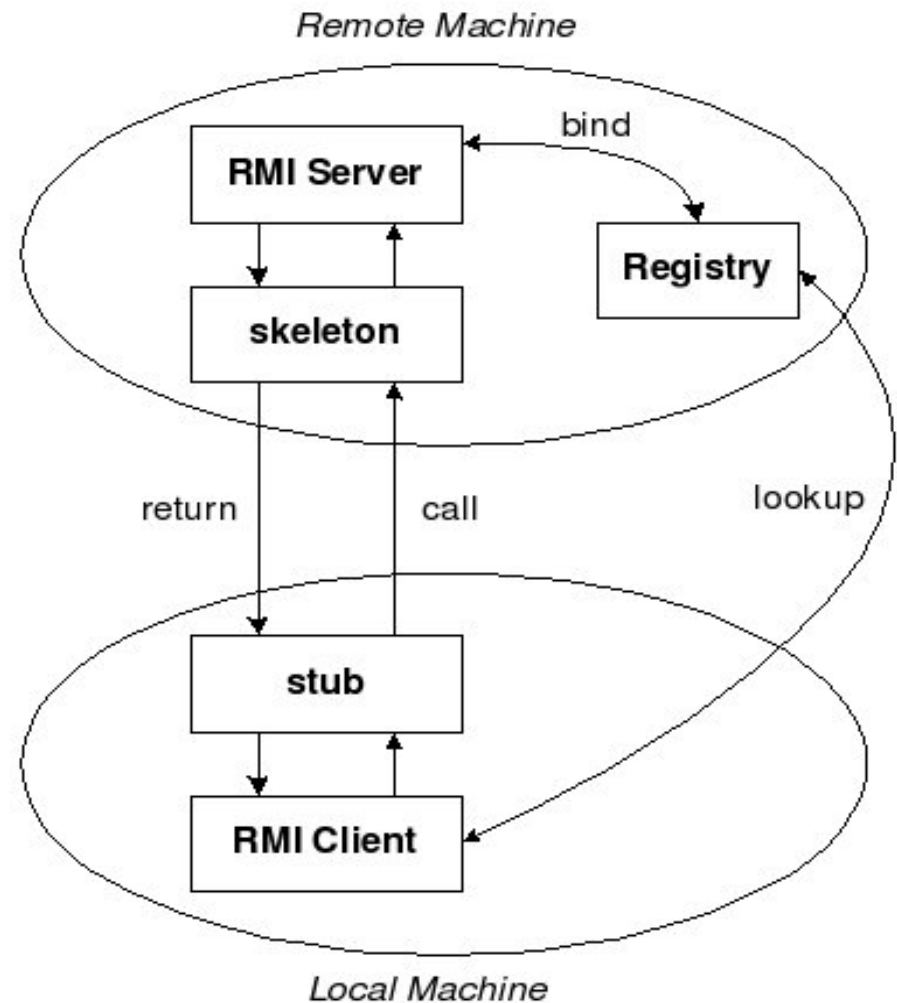
- A Implementação

- Deve estender:
`java.rmi.server.UnicastRemoteObject`
- Deve implementar a interface que declara os serviços.



RMI

- O Servidor é a classe que implementa a interface.
- Sua responsabilidade é:
 - Criar uma instância da classe Servidor
 - Registrar esta referência no registro de objetos RMI



- O Servidor de Registros RMI
 - É um serviço que aceita conexões na porta 1099 (default).
 - Nele registramos as referências dos objetos compartilhados e seus respectivos nomes.
 - O JDK vem com um servidor de registros RMI: rmiregistry
 - Também é possível criar um servidor utilizando a classe `java.rmi.registry.LocateRegistry` e seu método `createRegistry()`.

- O registro
 - Registramos uma referência de Objeto e seu respectivo nome com o método `bind()` de uma das classes abaixo:
 - `java.rmi.Naming`
 - `java.naming.InitialContext`

- `java.rmi.Naming`

- Pertence a especificação RMI

Ex.:

```
java.rmi.Naming.bind("nome", new ObjetoServer());
```

- nome é o nome que será associado ao objeto que implementou a interface remota.
 - ObjetoServer é uma instância do objeto que implementou a interface remota.

- `java.naming.InitialContext`
 - Pertence a especificação JNDI (Java Naming and Directory Interface)

Exemplo 1:

```
Properties env = new Properties();  
  
env.put(Context.INITIAL_CONTEXT_FACTORY,  
        "com.sun.jndi.rmi.registry.RegistryContextFactory");  
  
env.put(Context.PROVIDER_URL, "rmi://hostname:1099");  
  
Context ctx = new InitialContext(env);  
  
ctx.bind("nome", new ObjetoServer());
```

– Exemplo 2:

```
System.setProperty("java.naming.factory.initial",  
    "com.sun.jndi.rmi.registry.RegistryContextFactory");
```

```
System.setProperty("java.naming.provider.url", "rmi://hostname:1099");
```

```
Context ctx = new InitialContext();
```

```
ctx.bind("nome", new ObjetoServer());
```

- A diferença entre os dois exemplos ocorre na forma como as propriedades da JVM são configuradas.