PUC-SP

Java RMI (Remote Method Invocation)

Prof. Carlos Eduardo de Barros Paes

Departamento de Computação

PUC-SP

Introdução

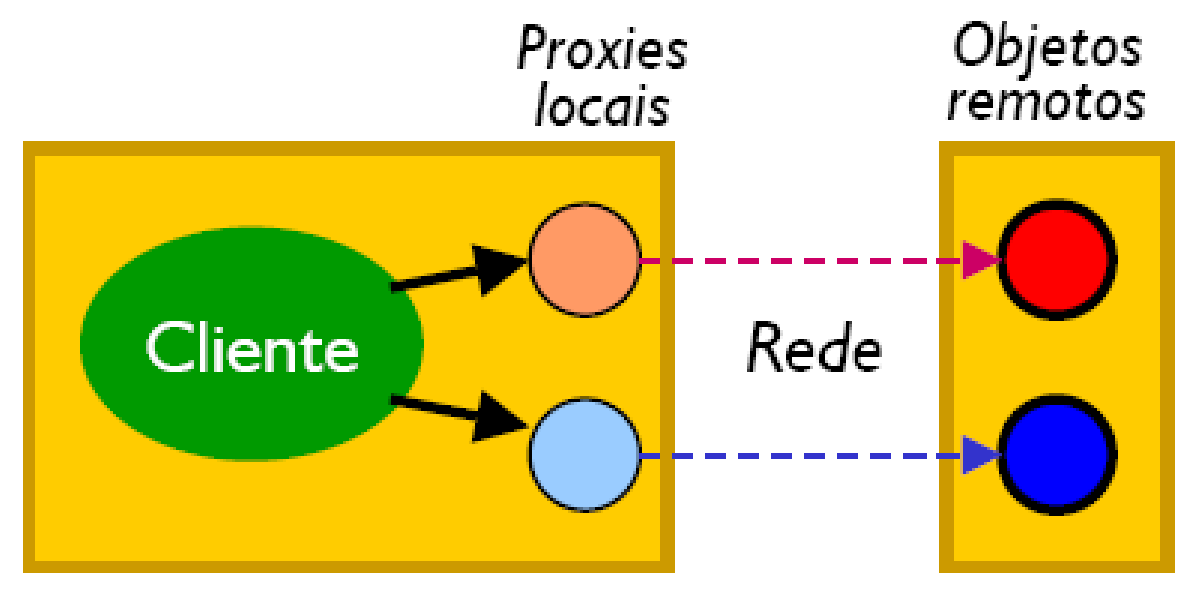
- Solução JAVA para Objetos Distribuídos
- Um objeto existe em uma máquina
- É possível se comunicar com esse objeto remotamente
  - Enviar mensagem para o objeto, através da chamada de métodos
  - Receber resultados do objeto

O que são Objetos Remotos

- Objetos remotos são objetos cujos métodos podem ser chamados remotamente, como se fossem locais
  - Cliente precisa, de alguma forma, localizar e obter uma instância do objeto remoto (geralmente através de um proxy intermediário gerado automaticamente)
  - Depois que o cliente tem a referência, faz chamadas nos métodos do objeto remoto (através do proxy) usando a mesma sintaxe que usaria se o objeto fosse local

O que são Objetos Remotos

- Objetos remotos abstraem toda a complexidade da comunicação em rede
  - Estende o paradigma OO
    além do domínio local
  - Torna a rede transparente

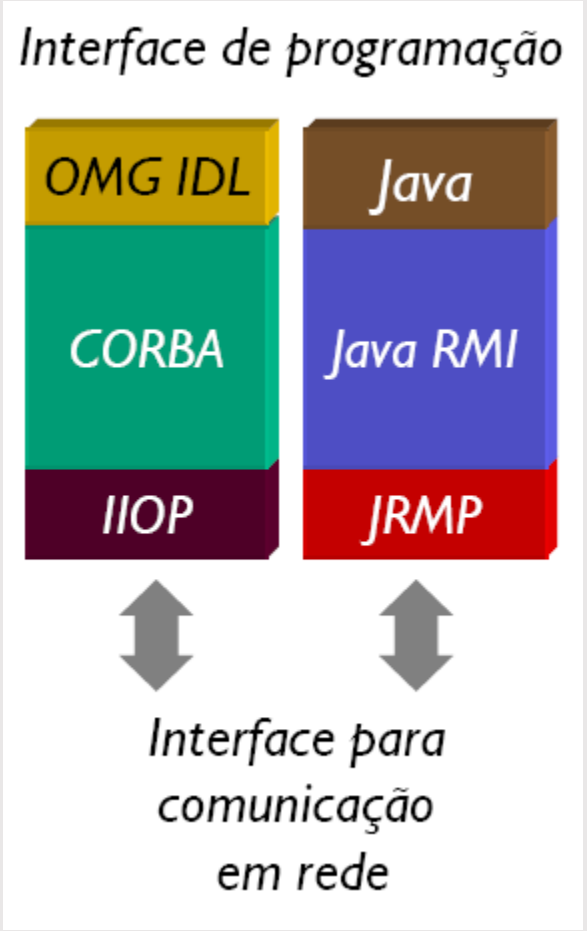


Como implementar

- Para usar objetos remotos, é preciso ter uma infraestrutura que cuide da geração das classes, comunicação e outros detalhes
- Há duas soluções disponíveis para implementar objetos remotos em Java
  - OMG **CORBA**: requer o uso, além de Java, da linguagem genérica OMG IDL, mas suporta integração com outras linguagens
  - Java **RMI**: para soluções 100% Java

Como implementar

- As duas soluções diferem principalmente na forma de implementação
- Em ambas, um programa cliente poderá chamar um método em um objeto remoto da mesma maneira como faz com um método de um objeto local

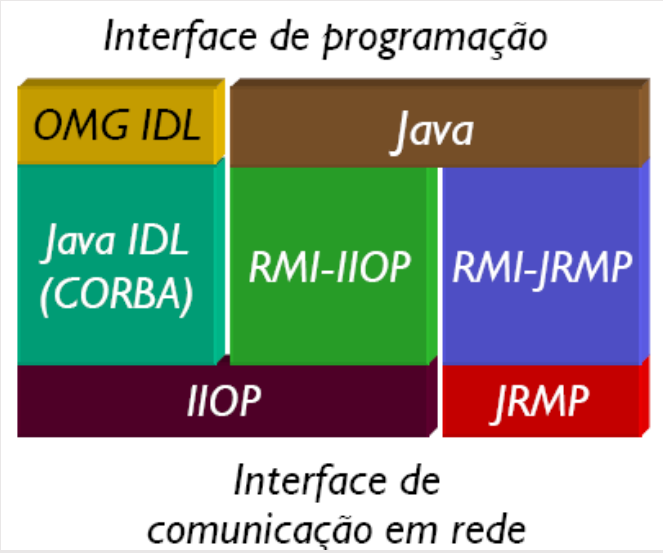


Objetos Remotos com RMI

- Java RMI (Remote Method Invocation) pode ser implementado usando protocolos e infra-estrutura próprios do Java (JRMP e RMI Registry) ou usando IIOP e ORBs, próprios do CORBA
- JRMP - Java Remote Method Protocol
  - Pacote **java.rmi** - RMI básico
  - Ideal para aplicações 100% Java.
- IIOP - Internet Inter-ORB Protocol
  - Pacote **javax.rmi** - RMI sobre IIOP
  - Ideal para ambientes heterogêneos.

Objetos Remotos com RMI

- A forma de desenvolvimento é similar
  - Há pequenas diferenças na geração da infra-estrutura (proxies) e registro de objetos
- RMI sobre IIOP permite programação Java RMI e comunicação em CORBA, viabilizando integração entre Java e outras linguagens sem a necessidade de aprender OMG IDL



RMI: Funcionamento

- Um objeto remoto previamente registrado é obtido, através de servidor de nomes especial: RMI Registry.
  - Permite que os objetos publicamente acessíveis através da rede sejam referenciados através de um nome.
- Serviço de nomes: classe java.rmi.Naming
  - Método Naming.lookup() consulta um servidor de nomes RMI e obtém uma instância de um objeto remoto
- Exemplo (jogo de batalha naval):

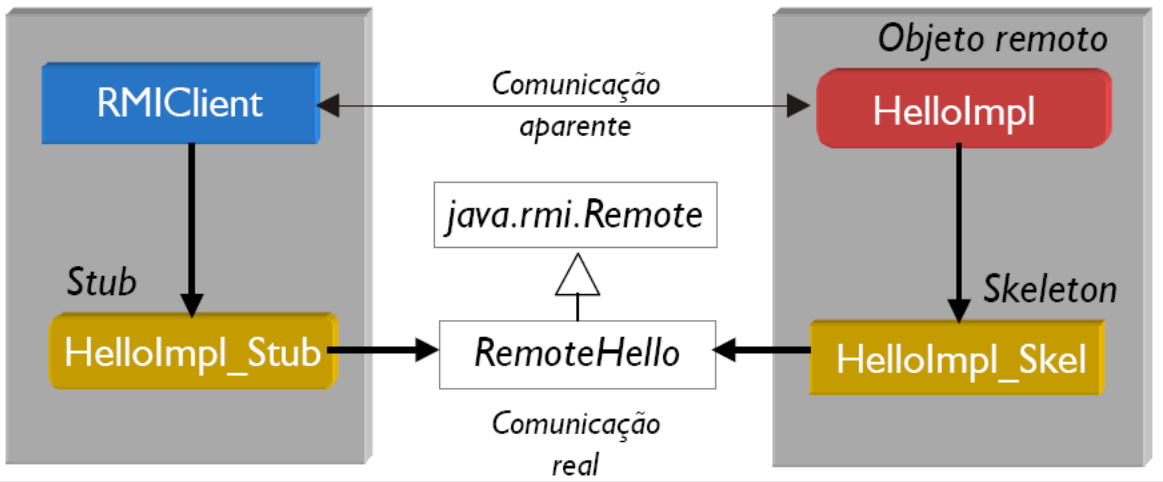
```
Territorio mar =
(Territorio)Naming.lookup("rmi://gamma/caspio");
```

- Agora é possível chamar métodos remotos de mar:

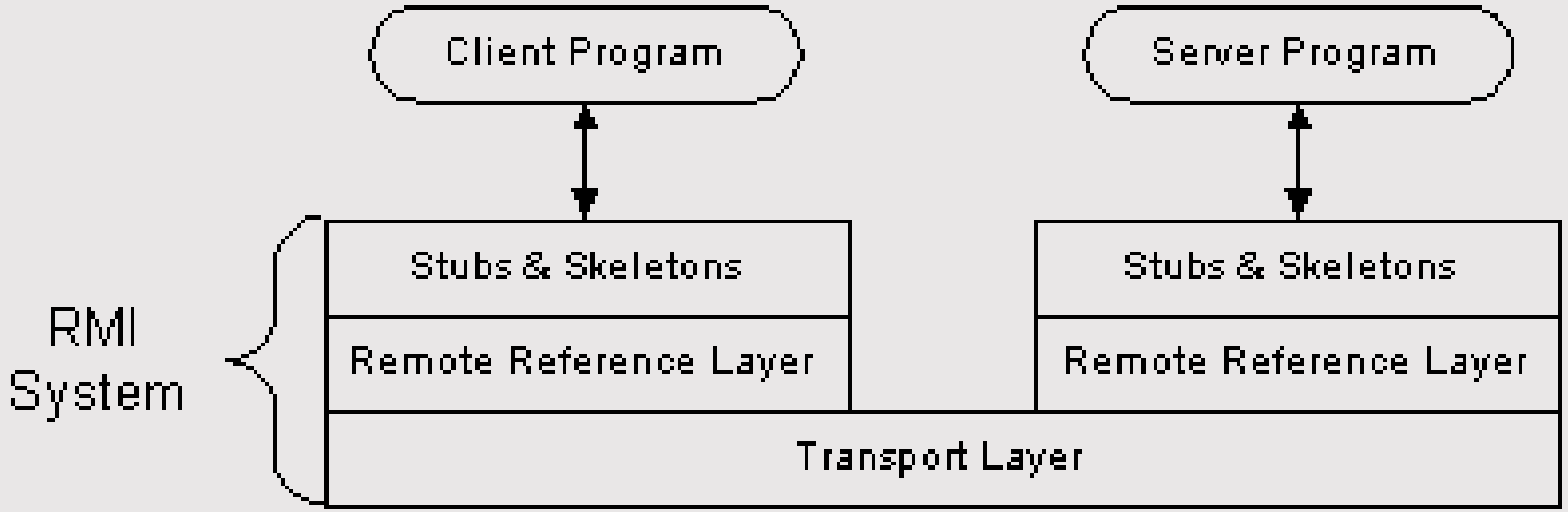
```
tentativa[i] = mar.atira("C", 9);
```

Arquitetura do RMI

- Uma aplicação distribuída com RMI tem acesso transparente ao objeto remoto através de sua Interface Remota
  - A "Interface Remota" é uma interface que estende java.rmi.Remote
  - A partir da Interface Remota e implementação do objeto remoto o sistema gera objetos (proxies) que realizam todas as tarefas necessárias para viabilizar a comunicação em rede



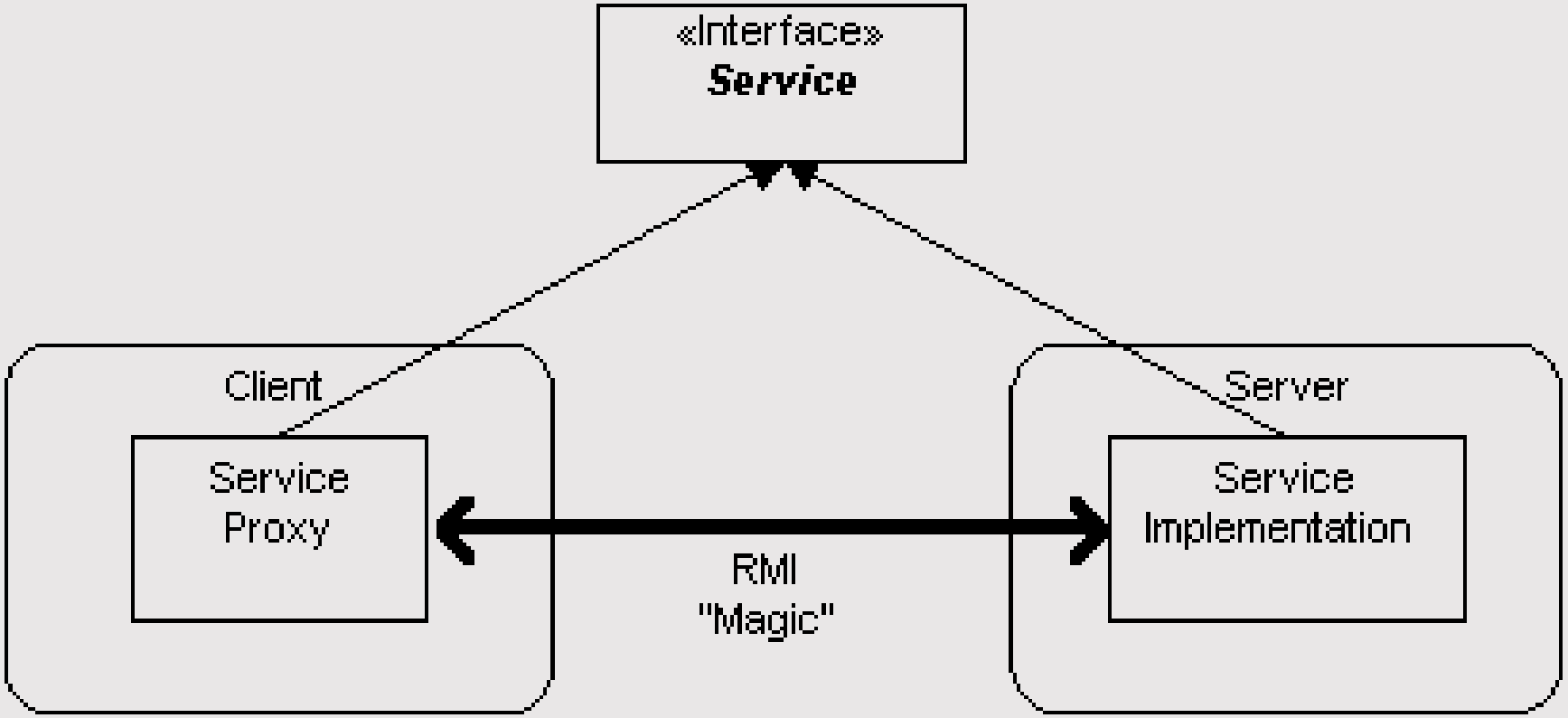
RMI Arquitetura em Camadas



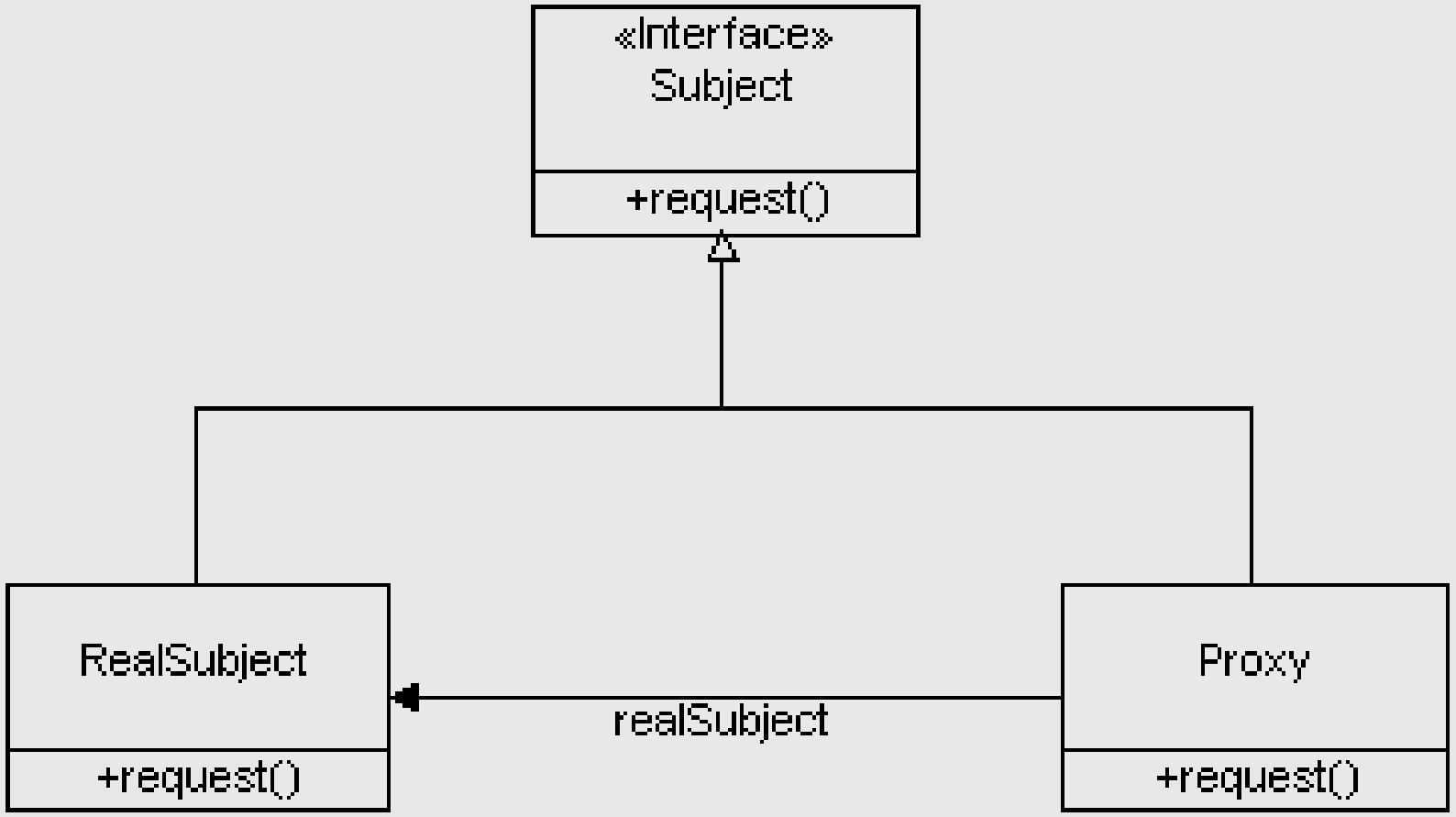
Padrões de projeto

- A implementação RMI é um exemplo do padrão de projeto chamado **Proxy**
- Proxy é uma solução para situações onde o objeto de interesse está inacessível diretamente, mas o cliente precisa operar em uma interface idêntica
  - A solução oferecida por Proxy é criar uma classe que tenha a mesma interface que o objeto de interesse (implemente a mesma interface Java) e que implemente, em seus métodos, a lógica de comunicação com o objeto inacessível.
- Em RMI, o proxy é o Stub gerado automaticamente pelo ambiente de desenvolvimento (rmic)

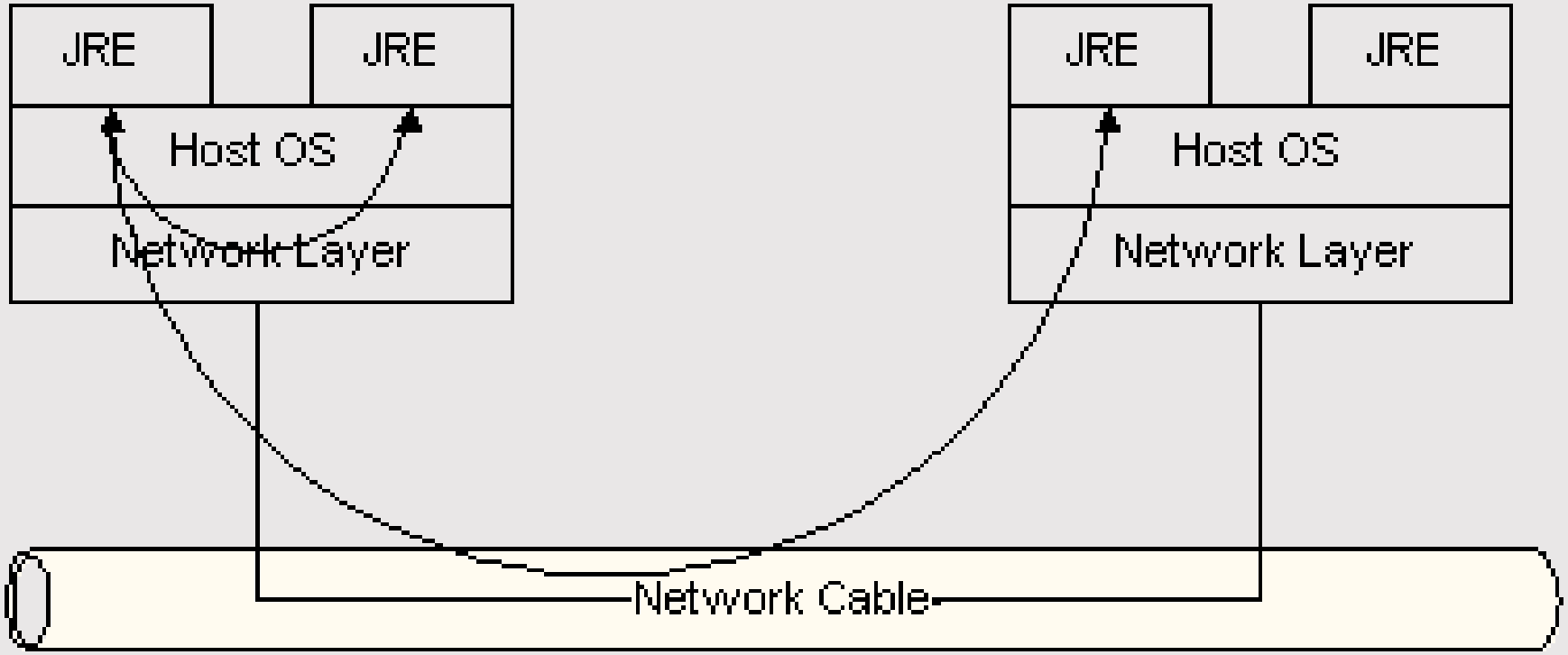
Interface Remota



Camadas Stub e Skeleton



Camada de Transporte



RMI em 10 passos

- *Passos básicos de criação de objetos distribuídos em Java*
  1. *Definir a interface*
  2. *Implementar os objetos remotos*
  3. *Implementar um servidor para os objetos*
  4. *Compilar os objetos remotos*
  5. *Gerar stubs e skeletons com rmic **
  6. *Escrever, compilar e instalar o(s) cliente(s)*
  7. *Instalar o stub no(s) cliente(s)*
  8. *Iniciar o RMI Registry no servidor*
  9. *Iniciar o servidor de objetos*
  10. *Iniciar os clientes informando o endereço do servidor.*

1. Definir a Interface Remota

- Declare todos os métodos que serão acessíveis remotamente em uma interface Java que estenda java.rmi.Remote.
  - Todos os métodos devem declarar throws java.rmi.RemoteException.
- Isto deve ser feito para cada objeto que será acessível através da rede.

```
import java.rmi.*;
public interface Mensagem extends Remote {
    public String getMensagem() throws RemoteException;
    public void setMensagem(String msg) throws
RemoteException;
}
```

2. Implementar Objetos Remotos

- Cada objeto remoto é uma classe que estende a classe java.rmi.server.UnicastRemoteObject e que implementa a interface remota criada no passo 1.
- Todos os métodos declaram causar java.rmi.RemoteException inclusive o construtor, mesmo que seja vazio.

```
import java.rmi.server.*;
import java.rmi.*;
public class MensagemImpl extends UnicastRemoteObject implements Mensagem {
    private String mensagem = "Inicial";
    public MensagemImpl() throws RemoteException {}
    public String getMensagem() throws RemoteException {
        return mensagem;
    }
    public void setMensagem(String msg) throws RemoteException {
        mensagem = msg;
    }
}
```

3. Estabelecer um Servidor

- Crie uma classe que
  a) Crie uma instância do objeto a ser servido e
  b) Registre-a (bind ou rebind) no serviço de nomes.

```
import java.rmi.*;
public class MensagemServer {
    public static void main(String[] args) throws RemoteException {
        Mensagem mens = new MensagemImpl();
        Naming.rebind("mensagens", mens);
        System.out.println("Servidor no ar.  Nome do objeto servido: "
                           + "mensagens" +"""");
    }
}
```

4. Compilar os objetos remotos

- Compile todas as interfaces e classes utilizadas para implementar as interfaces Remote

> javac Mensagem.java MensagemImpl.java

5. Gerar Stubs e Skeletons

- Use a ferramenta do J2SDK: **rmic**
- Será gerado um arquivo stub
  - **MensagemImpl_stub.class**

  e um arquivo skeleton
  - **MensagemImpl_skel.class**

  para cada objeto remoto (neste caso, apenas um)
- RMIC = **RMI C**ompiler
  - Use opção -keep se quiser manter código-fonte
  - Execute o rmic sobre as implementações do objeto remoto já compiladas:

> rmic MensagemImpl

6. Compilar e Instalar os Clientes

- Escreva uma classe cliente que localize o(s) objeto(s) no serviço de nomes (java.rmi.Naming)
  - a) Obtenha uma instância remota de cada objeto
  - b) Use o objeto, chamando seus métodos

```
import java.rmi.*;
public class MensagemClient {
    public static void main(String[] args) throws Exception {
        String hostname = args[0];
        String objeto = args[1];
        Object obj = Naming.lookup("rmi://" + hostname + "/"+ objeto);
        Mensagem mens = (Mensagem) obj;
        System.out.println("Mensagem recebida:  "+ mens.getMensagem());
        mens.setMensagem("Fulano esteve aqui!");
    }
}
```

7. Distribuir o Stub no(s) Cliente(s)

- Distribua o cliente para as máquinas-cliente. A distribuição deve conter
  - Classe(s) que implementa(m) o cliente (HelloMensagem.class)
  - Os stubs (HelloMensagem_stub.class)
  - As interfaces Remote (Mensagem.class)

- Em aplicações reais, os stubs podem ser mantidos no servidor
  - O cliente faz download do stub quando quiser usá-lo
  - Para isto é preciso definir algumas propriedades adicionais (omitidas no exemplo simples) como Codebase (CLASSPATH distribuído), SecurityManager e políticas de segurança (Policy)

8. Iniciar o RMI Registry no Servidor

- No Windows

  > start rmiregistry

- (O RMI Registry fica "calado" quando está rodando)
- Neste exemplo será preciso iniciar o RMIRegistry no diretório onde estão os stubs e interface Remote
  - Isto é para que o RMIRegistry veja o mesmo CLASSPATH que o resto da aplicação
  - Em aplicações RMI reais isto não é necessário, mas é preciso definir a propriedade java.rmi.server.codebase contendo os caminhos onde se pode localizar o código

Iniciar o servidor de objetos

- O servidor é uma aplicação executável que registra os objetos no RMIRegistry. Rode a aplicação:
  - **> java MensagemServer**

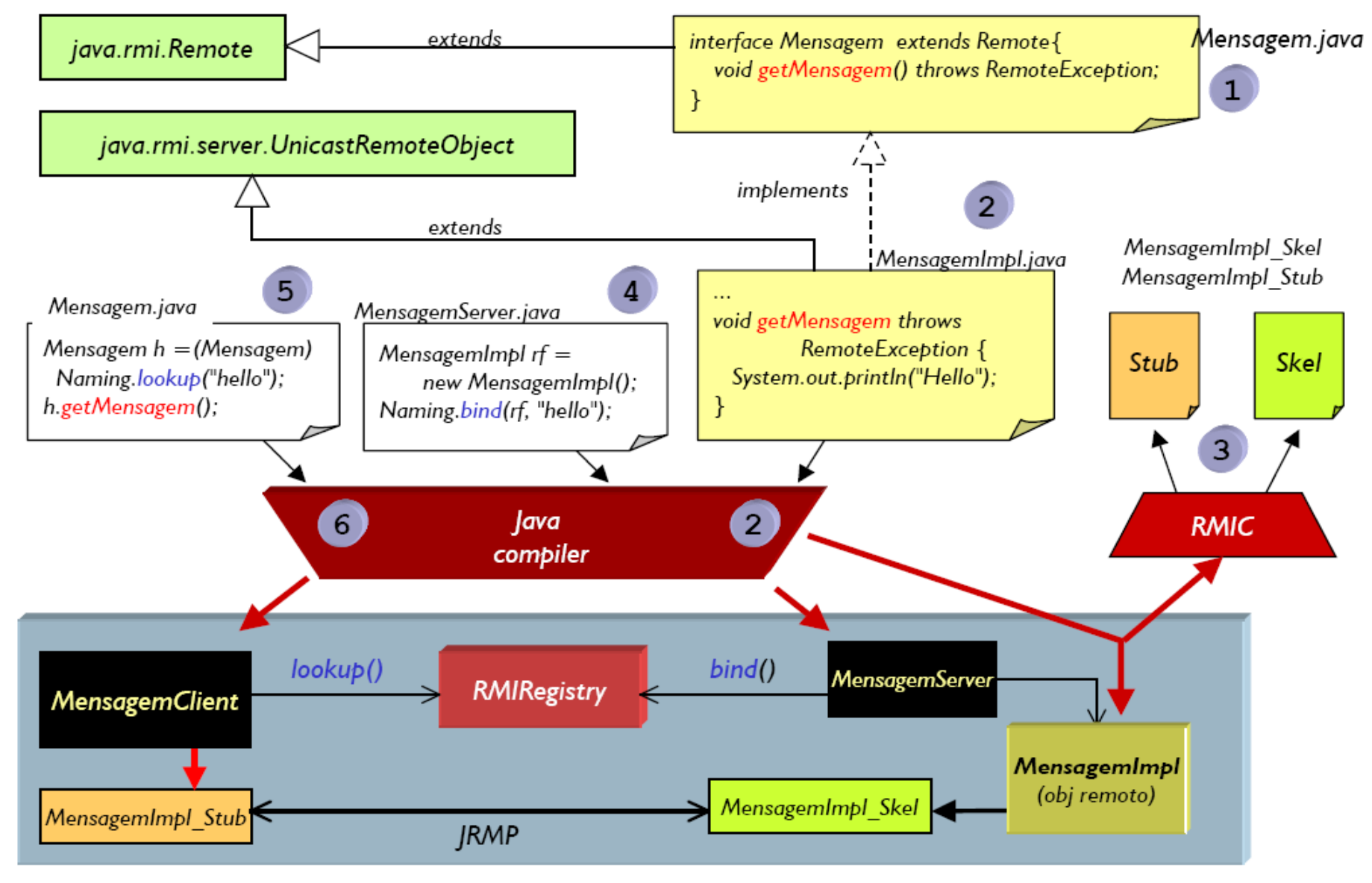
  Servidor no ar. Nome do objeto servido: mensagens
- Neste exemplo será preciso iniciar o servidor no diretório onde estão os stubs e interface Remote
  - Isto é para que o RMIRegistry veja o mesmo CLASSPATH que o resto da aplicação
  - Em aplicações RMI reais isto não é necessário, mas é preciso definir a propriedade java.rmi.server.codebase contendo os caminhos onde se pode localizar o código

10. Iniciar os Clientes

- *Rode o cliente*
  - *Informe o endereço do servidor e objetos a utilizar*

  > java MensagemClient maquina.com.br mensagens

Resumo


java.rmi.Remote
extends
interface Mensagem extends Remote{
void getMensagem() throws RemoteException;
}
Mensagem.java
1
java.rmi.server.UnicastRemoteObject
implements
2
extends
MensagemImpl.java
...
void getMensagem throws
RemoteException {
System.out.println("Hello");
}
MensagemImpl_Skel
MensagemImpl_Stub
Stub
Skel
3
RMIC
5
Mensagem.java
Mensagem h =(Mensagem)
Naming.lookup("hello");
h.getMensagem();
4
MensagemServer.java
MensagemImpl rf =
new MensagemImpl();
Naming.bind(rf, "hello");
6
Java
compiler
2
MensagemClient
lookup()
RMIRegistry
bind()
MensagemServer
MensagemImpl
(obj remoto)
MensagemImpl_Stub
JRMP
MensagemImpl_Skel

Parâmetros

- Tipos de dados primitivo → passagem por valor
- Objetos → passagem por valor
  - Objeto dever ser serializável
- Objetos Remotos → referência remota