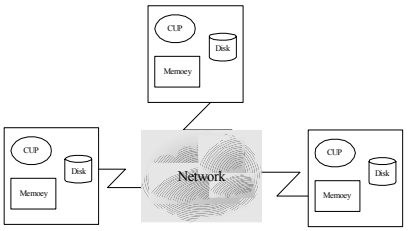
SISTEMAS DISTRIBUÍDOS



Remote Method Invocation (RMI)

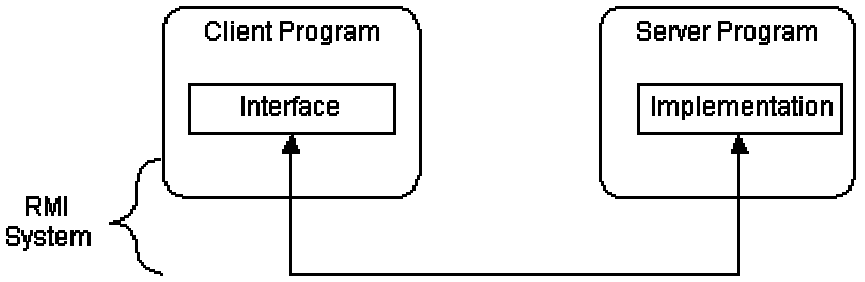
Introdução

- 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

Introdução

- Componentes necessários
  - Interface Remota
  - Classe Remota que implementa a interface
  - Stubs e Skeletons (gerados automaticamente)
  - Classe Cliente que usa o objeto remoto

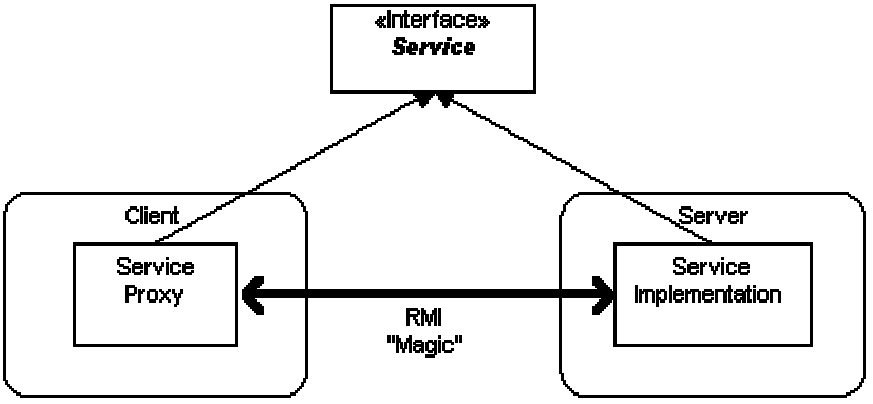
Introdução



Interface Remota

- Todo objeto para ser acessado remotamente deve possuir uma interface associada
- Características da interface (ECKEL, 1998)
  - Deve ser pública (por padrão uma interface é pública)
  - Deve estender a interface java.rmi.Remote
  - Cada método deve lançar uma exceção java.rmi.RemoteException
  - Todos objetos passados como argumento ou valor de retorno devem aparecer na interface remota

Interface Remota



Interface Remota

- Exemplo: Interface Hello.java

```
package examples;
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface Hello extends Remote {
   String sayHello() throws RemoteException;
}
```


Implementação da Interface

- Características do Servidor
  - deve possuir uma classe que estende UnicastRemoteObject
  - deve implementar a interface remota
  - a classe pode possuir vários métodos, entretanto somente os métodos declarados na interface são acessíveis remotamente
  - é necessário definir explicitamente o construtor, pois também deve lançar RemoteException

Implementação da Interface

- Características do Servidor
  - Deve ser criado um gerenciador de segurança que suporte RMI como o RMISecurityManager
  - Deve ser criada uma instância do objeto remoto
  - O objeto criado deve ser registrado através do método Naming.rebind

Implementação da Interface

- Exemplo: Classe HelloImpl.java

```
package examples;
import java.rmi.Naming;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class HelloImpl extends UnicastRemoteObject
      implements Hello {
      public HelloImpl() throws RemoteException {
         super();
      }
      public String sayHello() {
         return "Hello World!";
    }
```

Implementação da Interface

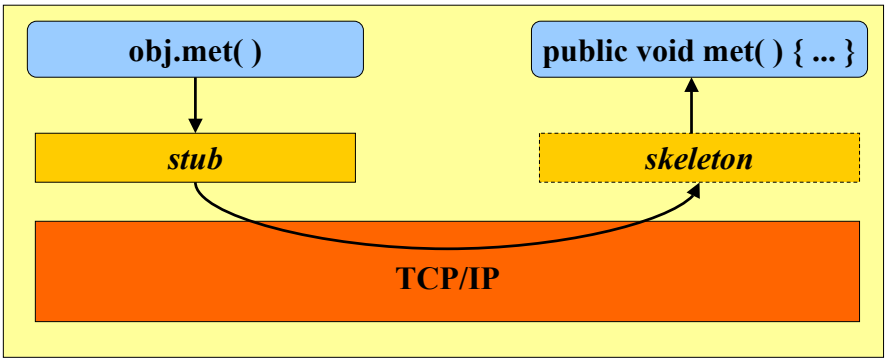
- Exemplo: Classe HelloImpl.java

```
public static void main(String args[]) {
    try {
            HelloImpl obj = new HelloImpl();
            // Associa o objeto com o nome "HelloServer"
            Naming.rebind("//localhost:2004/HelloServer", obj);
            System.out.println("HelloServer bound in registry");
         } catch (Exception e) {
        System.out.println("HelloImpl err: " + e.getMessage());
        e.printStackTrace();
       }
  }
}
```


Registro do Objeto Remoto

- É necessário registrar o objeto na máquina para acesso remoto
  - O processo rmiregistry deve ser iniciado (No Windows start rmiregistry).
  - Para o rmiregistry pode ser especificado como parâmetro uma porta. A porta padrão é 1099 (LocateRegistry.createRegistry(2004))
  - Devem ser criados os stubs e skeletons do objeto através do comando rmic

Registro do Objeto Remoto



compilador RMI (*rmic*) gera *stub* e *skeleton*

Registro do Objeto Remoto

- É necessário registrar o objeto na máquina para acesso remoto
  - Os stubs e skeletons fazem o empacotamento dos parâmetros que são objetos.
  - Todos parâmetros objetos devem ser serializáveis ou implementarem Remote
  - A classe remota deve ser executada. Mesmo com o fim da execução, pois o main termina, o objeto fica registrado

Registro do Objeto Remoto

- Seqüencia de comandos para o programa apresentado:
  - Compliar os arquivos Hello e HelloImpl
  - Executar o comando: rmic examples.HelloImpl
  - Executar o comando: start rmiregistry 2004

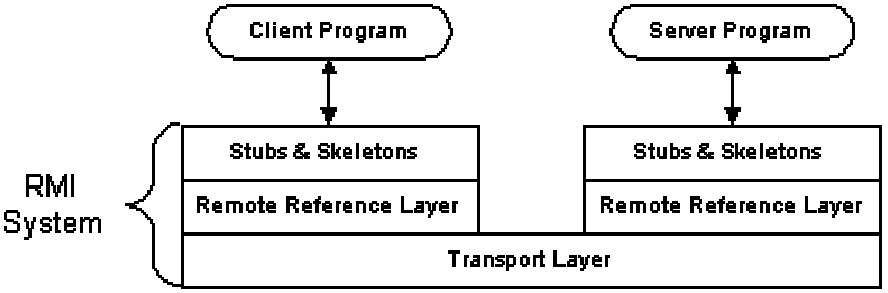
Objeto Cliente

- Exemplo: Classe HelloClient.java

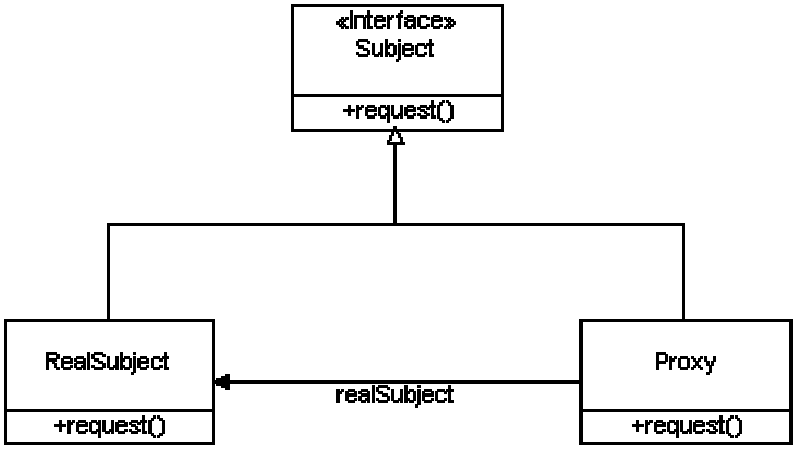
```
package examples;
import java.rmi.*;

public class HelloClient {
public static void main(String[] args) {
     String message = "blank";
     Hello obj = null;
     try {
              obj = (Hello)Naming.lookup("//localhost:2004/HelloServer");
              message = obj.sayHello();
     } catch (Exception e) {
     System.out.println("HelloCleint exception: " + e.getMessage());
     e.printStackTrace();
   }
   System.out.println(message);
     }
}
```

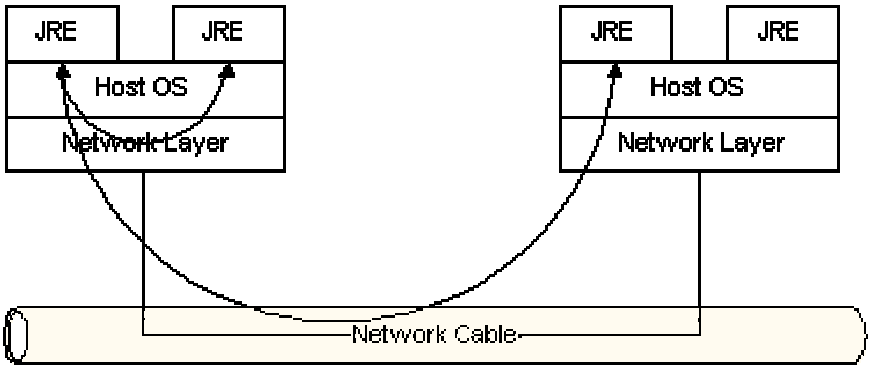
RMI Arquitetura em Camadas



Camadas Stub e Skeleton



Camada de Transporte



Parâmetros

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