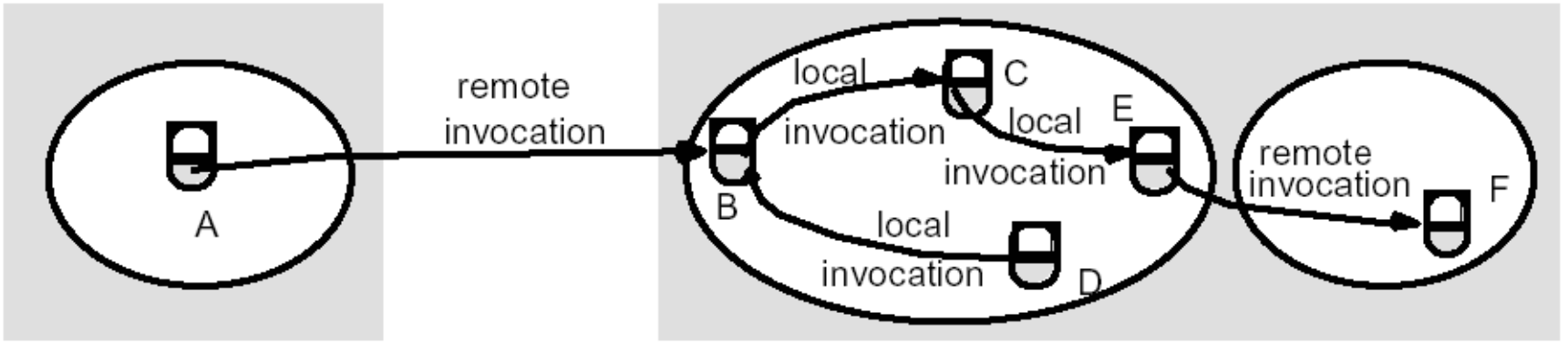
Invocação Remota de Métodos
Java RMI

Programação de Sistemas Distribuídos

Prof. MSc. Rodrigo Malara

Modelo de objetos distribuídos

- **Invocação de método remoto**: invocação de um método entre objetos localizados em diferentes processos, estejam estes na mesma máquina ou não.
- **Objetos remotos**: objetos que podem receber invocações remotas.



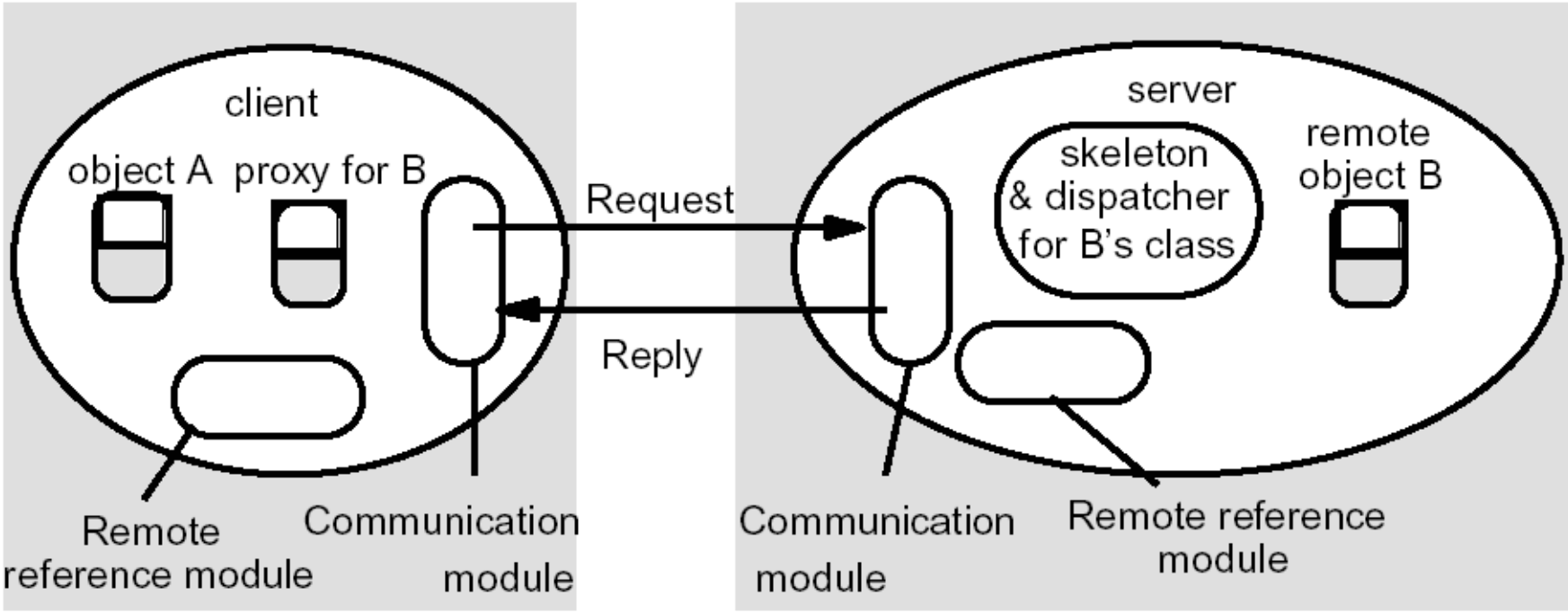
Modelo de objetos distribuídos

- **Referência a objeto remoto**: outros objetos podem invocar métodos de um objeto remoto somente se tiverem acesso a sua referência de objeto remoto
  - identificador único do objeto no sistema
  - pode ser utilizado como parâmetro ou resultado de uma invocação de método remoto
- **Interface remota**: todo objeto remoto possui uma interface remota que especifica qual dos métodos pode ser invocado remotamente
  - implementada pela classe do objeto remoto.

Considerações de projeto

- Transparência
  - A latência de uma chamada local é bastante menor que de uma chamada remota.
  - Chamadas remotas são mais vulneráveis a falhas do que chamadas locais.
  - As invocações de métodos remotos são iguais as invocações aos métodos locais, porém as diferenças entre os objetos remotos e locais devem estar expressos nas suas interfaces. (e.g. as classes de objetos remotos em Java implementam a interface *Remote*)

Implementação



Implementação

- Módulo de comunicação
  - protocolo request-reply
- Módulo de referência remota
  - realiza a tradução entre referências de objetos remotos e locais, e cria referências de objetos remotos
    - utiliza uma tabela de objetos remotos

Implementação

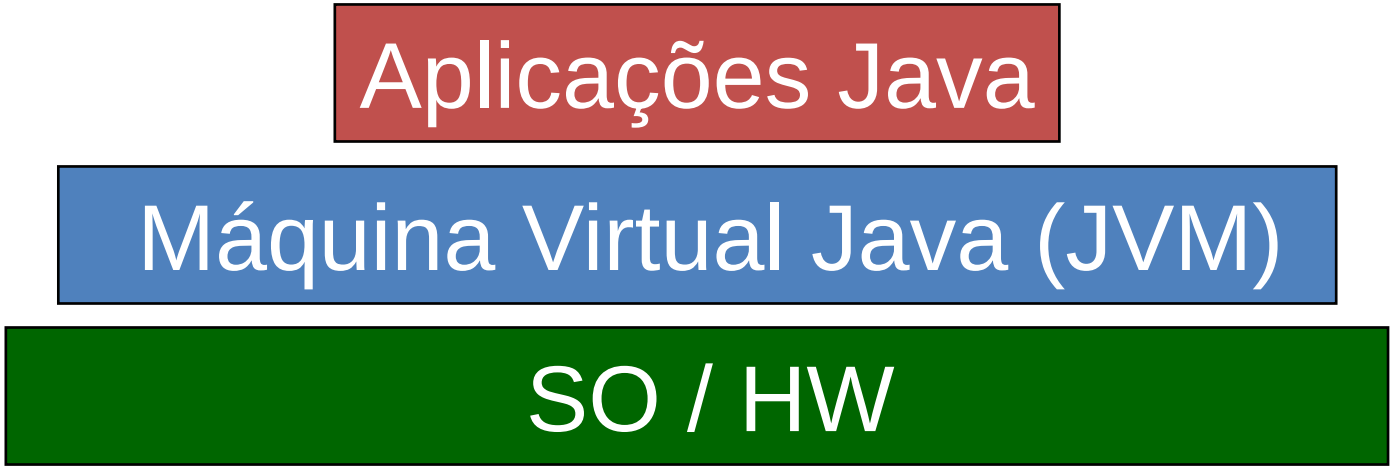
- **Proxy (localizado no cliente)**
  - marshaling de parâmetros, unmarshaling de resultados
- **Dispatcher (localizado no servidor)**
  - recebe a requisição do módulo de comunicação e chama o método indicado na mensagem (*methodId*)
- **Skeleton (localizado no servidor)**
  - unmarshaling dos parâmetros, marshaling dos resultados

Existe um proxy, um dispatcher e um skeleton para cada classe que representa um objeto remoto!

Implementação

- Geração das classes para proxies, dispatchers e skeletons
  - geradas pelo compilador de interfaces
- Threads do servidor
  - para cada invocação remota é disparada uma thread no servidor (concorrência)

Java



Java RMI

- RMI - Remote Method Invocation
- Java RMI - arquitetura de objetos distribuídos entre JVMs remotas
- Integra à linguagem Java o modelo de objetos distribuídos

Java RMI

- Utiliza mesma sintaxe para invocação de métodos de objetos locais ou remotos
- Todo objeto remoto implementa uma interface remota:
  - extende a interface java.rmi.Remote

    interface MyRemoteClass extends java.rmi.Remote
  - todos os métodos da interface devem prever a exceção *java.rmi.RemoteException* (cláusula *throws*) além das exceções próprias da aplicação

Java RMI - Serviço de nomes

- Acessado através da superclasse *java.rmi.Naming*
- Serviço RMIRegistry
- Através de nomes recupera referências para objetos
- Utiliza uma URL e porta TCP para cada servidor de nomes

Esse é o IDL do Java RMI

- IDL (Interface Definition Language)
- Sintaxe e semântica em Java
  - desvantagem: não suporta outras linguagens!

```
package br.com.uniara.sdpc;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface Calculadora extends Remote {
    public long somar(int a, int b) throws RemoteException;
    public long sub(int a, int b) throws RemoteException;
    public long mul(int a, int b) throws RemoteException;
    public long div(int a, int b) throws RemoteException;
}
```

Exemplo - objeto remoto

```
package br.com.uniara.sdpc;

import java.rmi.RemoteException;

public class CalculadoraImpl implements Calculadora{
    public long somar(int a, int b) throws RemoteException {
        return a + b;
    }
    public long sub(int a, int b) throws RemoteException {
        return a - b;
    }
    public long mul(int a, int b) throws RemoteException {
        return a * b;
    }
    public long div(int a, int b) throws RemoteException {
        return a / b;
    }
}
```

Exemplo - servidor

```
package br.com.uniara.sdpc;

import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;

public class Server {
  public static void main(String args[]) {
    try {
      Calculadora calc = new CalculadoraImpl(); // Obj remoto
      Calculadora skeleton = (Calculadora) UnicastRemoteObject
        .exportObject(calc, 0); // Skeleton do objeto remoto
      // Cria servidor RMI
      Registry registry = LocateRegistry.createRegistry(1099);
      // Adiciona skeleton ao servidor
      registry.bind("Hello", skeleton);
      System.err.println("Server ready");
    } catch (Exception e) {
      System.err.println("Server exception: " + e.toString());
      e.printStackTrace();
    }
  }
}
```

Exemplo - cliente

```
package br.com.uniara.sdpc;

import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import javax.swing.JOptionPane;

public class Client {
  public static void main(String[] args) {

    String host = (args.length < 1) ? null : args[0];
    try {
      Registry registry = LocateRegistry.getRegistry(host);
      Calculadora stub = (Calculadora) registry.lookup("Hello");

      Integer a = Integer.parseInt(JOptionPane.showInputDialog("Entre a"));
      Integer b = Integer.parseInt(JOptionPane.showInputDialog("Entre b"));

      System.out.println(" a + b = " + stub.somar(a, b));
      System.out.println(" a - b = " + stub.sub(a, b));
      System.out.println(" a * b = " + stub.mul(a, b));
      System.out.println(" a / b = " + stub.div(a, b));

    } catch (Exception e) {
      System.err.println("Client exception: " + e.toString());
      e.printStackTrace();
    }
  }
}
```

Java RMI

- *rmiregistry* (B*inder*)
  - deve ser executado em toda máquina que hospeda objetos remotos
  - mantém uma tabela para mapeamento da representação textual dos objetos aos objetos localizados na máquina
  - Em Java 5 não precisa mais ser executado a parte

Desenvolvimento usando Java RMI

- Passos
  - Projetar e implementar os componentes da aplicação distribuída
    - definição das interfaces remotas
    - implementação dos objetos remotos
    - implementação do servidor e clientes
  - Iniciar a aplicação
    - iniciar o servidor e cliente(s)

Exercício 1

- Um programa cliente usa RMI para conectar em um objeto remoto que recebe um número de conta corrente e agencia e retorna um objeto populado com os dados da conta corrente.

Passos:

1. Criar a classe Conta (1a) com os atributos conta, agencia, nome e saldo. (Obs não utilizar tipos primitivos)
2. Criar uma Interface ContaService (2o) contendo 1 método:

```
Conta getConta(String conta, String agencia)
```

3. Criar uma classe ContaServiceImpl (3a) que implemente ContaService e que implemente o método getConta acima, criando e retornando sempre o mesmo objeto (mockado)
4. Criar uma classe (4a) que executará o ContaServiceImpl no servidor
5. Criar uma classe cliente (5a) que invoque o método remoto e exiba os dados do objeto retornado.

Exercício 1 (cont.)

4. Criar uma classe que será executada e executará o ContaServiceImpl como servidor
5. Criar uma classe cliente que invoque o método remoto.

Exercício 2

- Modificar o exercício 1
  - Criar um HashMap com sendo que o número da conta é a chave e o conteúdo é o objeto conta
  - Crie um construtor para a classe ContaServiceImpl e nele inclua alguns clients para testes no HashMap.
  - Adicionar código que permita consultar se uma conta existe no HashMap e caso exista, retornar para o cliente
  - Criar uma classe chamada ContaNaoEncontradaException que extenda de RemoteException
    - Caso não encontre o cliente no Hashmap o método getConta deve lançar essa exceção