Seminář Java

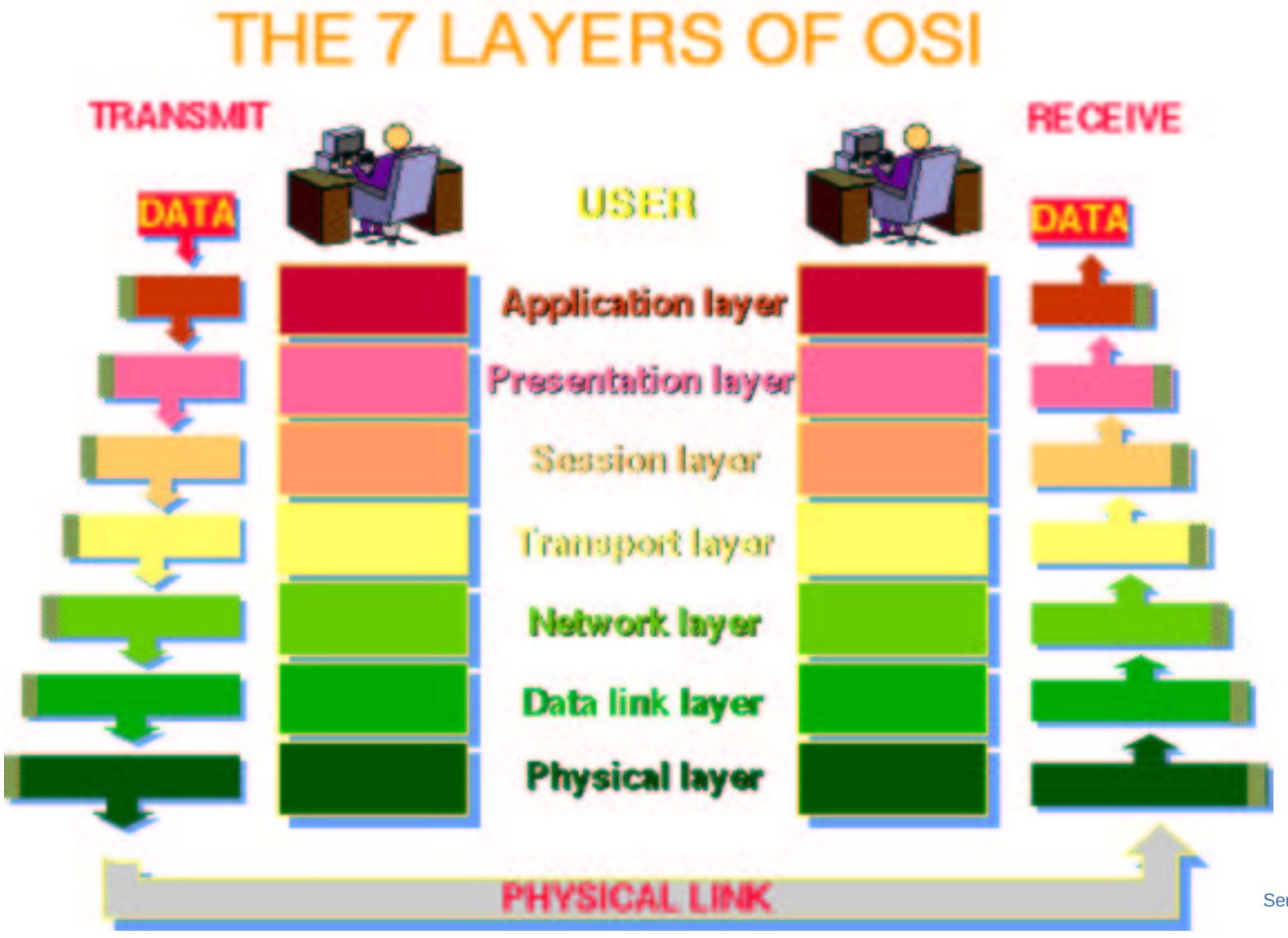
X

Obsah

- Java a sítě
- UDP, TCP
- Sokety
- HTTP
- RMI

Model ISO/OSI

Open System Interconnection



Model ISO/OSI

Aplikační

- podpora end-user procesů (aplikace ftp, http, ...)

Prezentační

- forma přenosu dat (transformace, šifrování, ...)

Relační

- řízení spojení mezi aplikačními procesy

Transportní

- spojení, transparentní přenos dat (pakety, TCP, UDP)

Síťová

- směrování, adresování (IP, ...)

Linková

- pakety $\Rightarrow$ rámce, řízení toku dat na přenosovém médiu

Fyzická

- hardware, fyzické propojení (el. impulsy, signály, ...)

Protokoly

IP (Internet Protocol)

- síťová vrstva
- adresování (IP adresa)
- správné pořadí paketů

TCP (Transmission Control Protocol)

- transportní vrstva
- spojově orientovaný (záruka správného přenosu)

UDP (User Datagram Protocol)

- transportní vrstva
- rychlejší než TCP
- nespolehlivý ("pošta")

Porty

Připojení k síti

- obvykle jedno fyzické připojení
- všechna data procházejí tímto připojením
- problém přiřazení konkrétních dat konkrétní aplikaci

Porty

- protokoly TCP a UDP mapují data na jednotlivé procesy podle portů
- port je 16bitové číslo
- 0 – 1023 vyhraženo pro standardní služby (http, ssh, ...)
- adresace dat = adresa stroje (IP) + port

Java a sítě

Balíček `java.net`

- Podpora komunikace s TCP
  - `URL`
  - `URLConnection`
  - `Socket`
  - `ServerSocket`
- Podpora komunikace s UDP
  - `DatagramPacket`
  - `DatagramSocket`
  - `MulticastSocket`

Java a sítě

`java.net.InetAddress`

- reprezentace IP adresy
- IPv4 (32 bits)
- IPv6 (128 bits)
- unicast i multicast
- ...

Třída `java.net.URL`

URL (Uniform Resource Locator)

- adresa zdroje v internetu
- adresa má dvě základní části
  - identifikátor protokolu (*`http, ftp, ...`*)
  - název zdroje (*`hostname, filename, port, ...`*)
- `http://perchta.fit.vutbr.cz:8080/java`

Vytvoření spojení (konstruktory)

- "write-once" objekty
- výjimka `MalformedURLException`
- `new URL("http://perchta.fit.vutbr.cz:8080/java")`
- `new URL("http", "perchta.fit.vutbr.cz", 8080, "java")`
- `...`

Práce s URL

Získání informací o URL

- `getProtocol`, `getHost`
- `getPort`, `getFile`, `...`
- ne každá URL má všechny části

Ukázka:

```
URL aURL = new URL("http", "perchta.fit.vutbr.cz", 8080,
                                                   "java");

System.out.println("protocol = " + aURL.getProtocol());
System.out.println("host = " + aURL.getHost());
System.out.println("filename = " + aURL.getFile());
System.out.println("port = " + aURL.getPort());
System.out.println("ref = " + aURL.getRef());
```

Práce s URL

Přímé čtení

- `openStream()` – otevře spojení a vrátí objekt třídy `InputStream`
- čtení ze streamu

Ukázka:

```
URL yahoo = new URL("http://www.yahoo.com");
BufferedReader in = new BufferedReader(
        new InputStreamReader(yahoo.openStream()));

String inputline;
while ((inputline = in.readLine()) != null)
    System.out.println(inputline);

in.close();
```

Práce s URL

Vytvoření spojení

- metoda `openConnection`
- komunikační linka mezi procesem a URL
- po úspěšném navázání spojení $\Rightarrow$ objekt třídy `URLConnection`
- čtení pomocí `URLConnection` = přímé čtení
- zápis pomocí `URLConnection` = zápis dat na server
  - formuláře, ...
  - na straně serveru obvykle cgi-skript
- postup:
  - vytvoření URL
  - otevření spojení
  - získání proudu
  - zápis/čtení do/z proudu
  - uzavření proudu

Práce s URL

Vytvoření spojení – čtení

```
URL yahoo = new URL("http://www.yahoo.com");
URLConnection connection = yahoo.openConnection();
BufferedReader in = new BufferedReader(
        new InputStreamReader(connection.getInputStream()));

String inputline;
while ((inputline = in.readLine()) != null)
    System.out.println(inputline);

in.close();
```

Práce s URL

Vytvoření spojení – zápis

- reverze řetězce:
  `http://java.sun.com/cgi-bin/backwards`
- formát: *`string="řetězec"`*

```
String str = "Retezec";

URL url = new URL("http://java.sun.com/cgi-bin/backwards");
URLConnection connection = url.openConnection();

connection.setDoOutput(true); // směr komunikace
PrintWriter out = new PrintWriter(
        connection.getOutputStream());

out.println("string=" + str);
out.close();
```

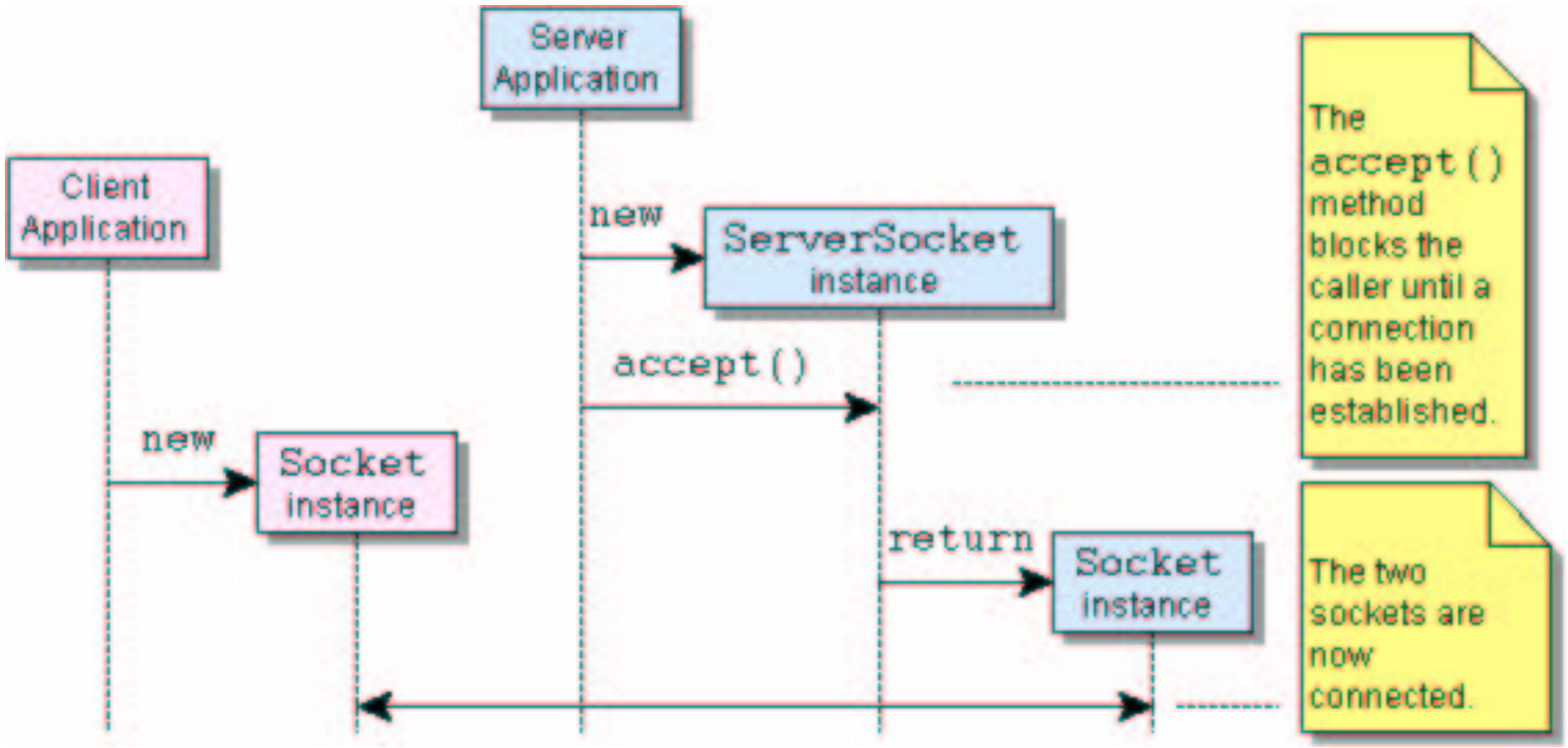
Sokety

Sokety

- komunikace typu klient – server
  - server čeká na žádost (připojení) klienta
  - klienti se připojují na server
  - server vytváří spojení
  - klient i server zasílají/čtou data
- vytvořené spojení
  - soket na straně klienta
  - soket na straně serveru
  - komunikace prostřednictvím proudů

`java.sun.com/j2se/1.5.0/docs/guide/net/overview/overview.html`

Sokety


Server Application
Client Application
new
ServerSocket instance
accept()
new
Socket instance
return
Socket instance
The accept() method blocks the caller until a connection has been established.
The two sockets are now connected.

Třída `java.net.Socket`

`java.net.Socket`

- reprezentuje jednu stranu komunikace
- soket na straně klienta
  - žádost o vytvoření spojení se serverem
  - `new Socket(String host, int port)`
  - `new Socket(InetAddress address, int port)`
- čtení ze soketu
  - získání vstupního proudu
  - `InputStream getInputStream()`
- zápis do soketu
  - získání výstupního proudu
  - `OutputStream getOutputStream()`

Třída `java.net.ServerSocket`

`java.net.ServerSocket`

- reprezentuje server
- není potomek třídy `Socket`!
- po připojení klienta vytváří objekt třídy `Socket`
- vytvoření serveru
  - `new ServerSocket(int port)`
  - `port` na kterém server naslouchá
- naslouchání spojení
  - metoda `accept()`
  - blokující operace
  - vytváří objekt třídy `Socket`

Sokety

Metody

- `getPort()`
- `getLocalPort()`
- `getInetAddress()`
- `getLocalAddress()`
- `...`

Sokety: ukázka aplikace

Server

```
ServerSocket ss = new ServerSocket(9000);
Socket s = ss.accept();

BufferedReader in = new BufferedReader(
       new InputStreamReader(s.getInputStream()));
PrintStream out = new PrintStream(s.getOutputStream());

System.out.println(in.readLine());
out.print("juchuu");
out.flush();

out.close();
in.close()
s.close();
ss.close();
```

Sokety: ukázka aplikace

Klient

```
try {
    s = new Socket("localhost", 9000);
    System.out.println("Client: new socket port " +
                                        s.getLocalPort());

    out = new PrintStream(s.getOutputStream());
    out.print("juchuu");
    out.flush();

    out.close();
    s.close();
}
catch (UnknownHostException ex) {}
catch (java.io.IOException ex) {}
```

Sokety: ukázka aplikace

Problém uvedené ukázky

- obslouží pouze jednoho klienta a pak skončí
- $\Rightarrow$ nekonečná (podmíněná) smyčka

```
ServerSocket ss;
try {
    ss = new ServerSocket(9000);
//  while(true) {
    for (int conns = 0; conns < 1; conns++) {
        Socket s = ss.accept();
        // obsluha klienta s
    }
    ss.close();
}
catch (java.io.IOException ex) { ex.printStackTrace();}
```

Sokety: ukázka aplikace

Problém uvedené ukázky

- při obsluze klienta nemůže obsloužit další
- souběžné obsloužení klientů
- $\Rightarrow$ vlákna

```
ServerSocket ss;
try {
    ss = new ServerSocket(9000);
//  while(true) {
    for (int conns = 0; conns < 1; conns++) {
        Socket s = ss.accept();
        (new WebClientThread(s)).start();
    }
    ss.close();
}
catch (java.io.IOException ex) { ex.printStackTrace();}
```

Sokety: ukázka aplikace

... pokračování

```
class ClientThread extends Thread {
    Socket s;

    public ClientThread(Socket s) {
        super();
        this.s = s;
    }

    public void run() {
        try {
            // obsluha klienta s
            s.close();
        } catch (java.io.IOException ex) {
            ex.printStackTrace();
        }
    }
}
```

Implementace jednoduchého WWW serveru

```
import java.net.*;
import java.io.*;

public class WebServer {
    public static void main(String[] argv) {
        new WebServer().start();
    }
    public void start() {
        try {
            ServerSocket ss = new ServerSocket(9000);
            for (int conns = 0; conns < 1; conns++) {
                Socket s = ss.accept();
                (new WebClientThread(s)).start();
            }
            ss.close();
        }
        catch (java.io.IOException ex) { ... }
    }
}
```

Implementace jednoduchého WWW serveru

```
class WebClientThread extends Thread {
   Socket s;
   public WebClientThread(Socket s) {
      super();
      this.s = s;
   }
   public void run() {
      try {
         ...
         PrintStream out = new PrintStream(
                                   s.getOutputStream());
         out.print("<html>");
         ...
         out.print("<p>Time: " +
                   (new java.util.Date()).toString());
         ...
      } catch (java.io.IOException ex) { ... }
   }
}
```

Java RMI

RMI

- Remote Method Invocation
- objekt běžící na jednom JVM může volat metodu objektu bežícího na jiném JVM
- protokol komunikace je oddělen od aplikace
- podobné ale jednodušší než CORBA

Klient – server

- server registruje vzdálené objekty
- klient se dotazuje serveru na vzdálené objekty
- klient získává reference na vzdálené objekty $\Rightarrow$ jim zasílá zprávy
- klient se dotazuje serveru na vzdálené objekty

Java RMI

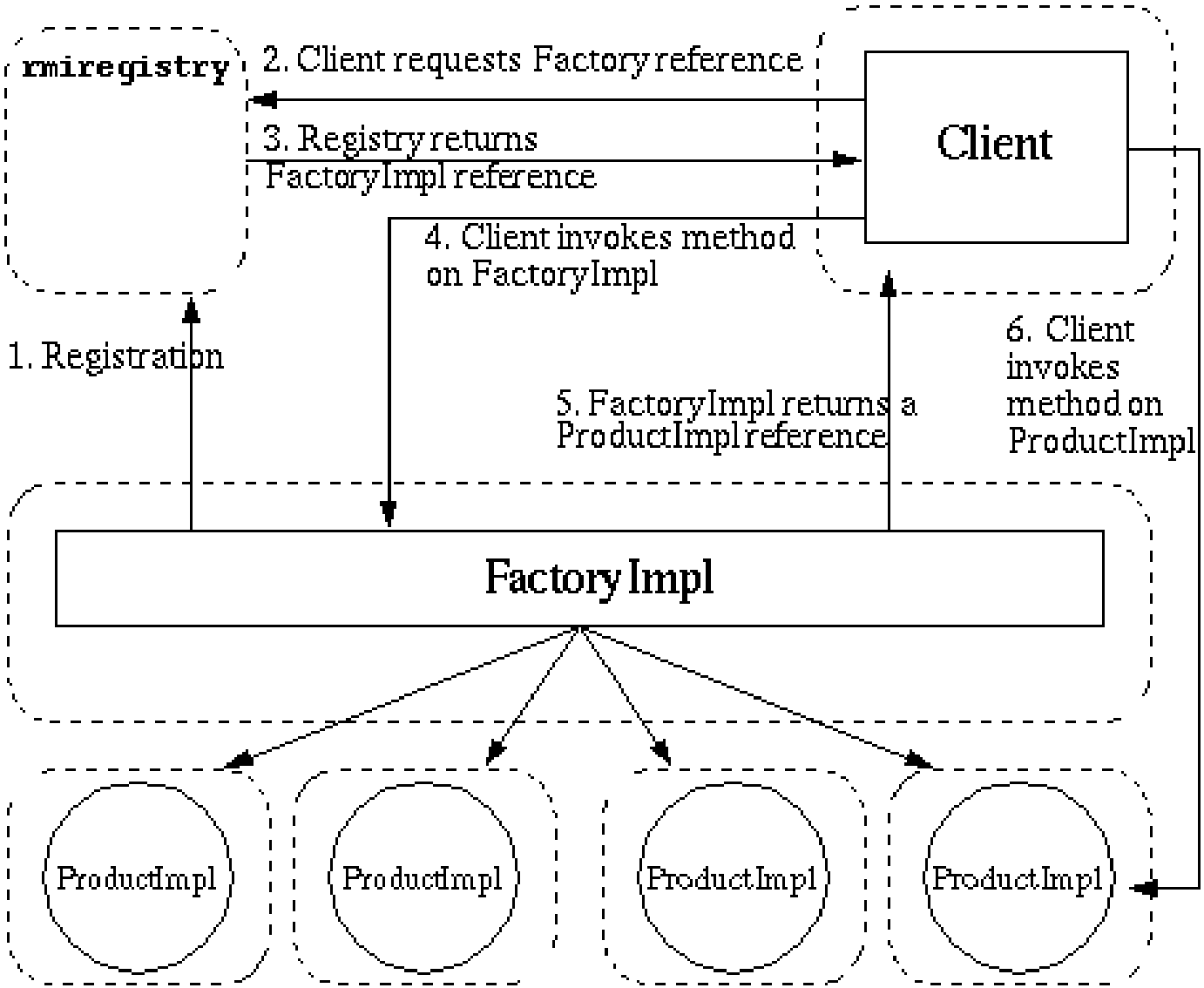
Skeleton

- entita na straně serveru
- reprezentuje vzdálený objekt
- provádí volání metod, serializaci/deserializaci

Stub

- lokální reprezentace vzdáleného objektu
- *ne jeho kopie!*
- provádí volání vzdáleného objektu, serializaci/deserializaci

Java RMI


rmiregistry
2. Client requests Factory reference
3. Registry returns FactoryImpl reference
Client
4. Client invokes method on FactoryImpl
1. Registration
5. FactoryImpl returns a ProductImpl reference
6. Client invokes method on ProductImpl
FactoryImpl
ProductImpl
ProductImpl
ProductImpl
ProductImpl

Java RMI: vzdálené objekty

Rozhraní `java.rmi.Remote`

- definuje vzdálené metody vzdáleného objektu
- každá vzdálená metoda musí deklarovat výjimku `RemoteException` v klauzuli `throws`

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

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

Java RMI: server

- třída `Server` implementuje rozhraní `Hello` $\Rightarrow$ vzdálený objekt
- pouze metody rozhraní mohou být vzdáleně volány

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

public class Server implements Hello {

    public Server() {}

    public String sayHello() {
        return "Hello, world!";
    }
    ...
```

Java RMI: server

- objekt je potřeba exportovat (vytvoření stub)
- stub je potřeba registrovat

```
public static void main(String args[]) {
    try {
        Server obj = new Server();
        Hello stub = (Hello)
                UnicastRemoteObject.exportObject(obj, 0);

        // Bind the remote object's stub in the registry
        Registry registry = LocateRegistry.getRegistry();
        registry.bind("Hello", stub);
    } catch (Exception e) {
        System.err.println("Server exception: " +
                                        e.toString());
        e.printStackTrace();
    }
}
```

Java RMI: klient

- musí se připojit ke vzdálenému serveru
- dotazuje se na registrované objekty

```
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class Client {
    public static void main(String[] args) {
        String host = (args.length < 1) ? null : args[0];
        try {
            Registry registry =
                            LocateRegistry.getRegistry(host);
            Hello stub = (Hello) registry.lookup("Hello");
            String response = stub.sayHello();
            System.out.println("response: " + response);
        } catch (Exception e) { ... }
    }
}
```

Java RMI: běh

Server

- kompilace
- nastavení `CLASSPATH`
- spuštění Java RMI registry
- spuštění serveru

```
export CLASSPATH="$CLASSPATH:classDir"
rmiregistry &
java -classpath classDir Server &
```

Klient

- spuštění klienta

```
java -classpath classDir Client
```

Poznámka: `rmic`

Java RMI: modifikace

- vzdálená metoda `getString()` bude mít argument
- třída argumentu musí implementovat rozhraní `Serializable`

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

public class Task implements Serializable {
    private String str = "task";
    public String getString() {
        return str;
    }
}
```

Java RMI: modifikace

- modifikace serveru a klienta

```
Server:
public String sayHello(Task task) {
    return "Hello, world! " + task.getString();
}
```

```
Klient:
Hello stub = (Hello) registry.lookup("Hello");
String response = stub.sayHello(new Task());
System.out.println("response: " + response);
```

Zdroje informací

http://www.javaworld.com

http://java.sun.com/j2se/1.5.0/docs/index.html

http://java.sun.com/j2se/1.5.0/docs/guide/rmi/index.html

http://java.sun.com/docs/books/tutorial/rmi/TOC.html

http://www.javacoffeebreak.com/articles/rmi_corba/

http://my.execpc.com/ gopalan/misc/compare.html

http://www.webopedia.com/quick_ref/OSI_Layers.asp