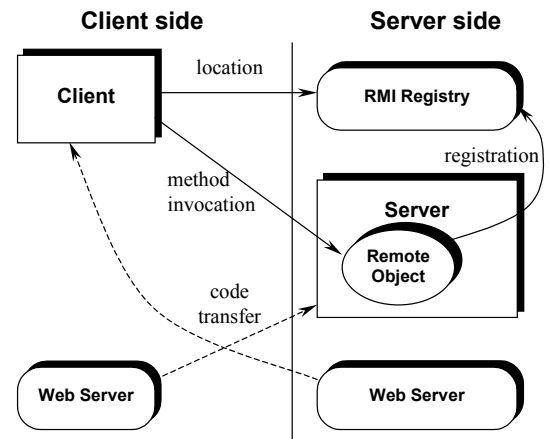


## Java RMI

### Etapy tworzenia aplikacji rozproszonej w środowisku Java RMI

1. Zdefiniowanie i implementacja odpowiednich klas (w szczególności klas dla obiektów dostępnych zdalnie)
  - ☞ zdefiniowanie interfejsu pochodnego od Remote
  - ☞ zdefiniowanie klasy wywiedzionej z klasy `java.rmi.server.UnicastRemoteObject`, implementującej interfejs pochodny od Remote
2. Kompilacja źródeł (`javac`, `rmic`)
  - ☞ `javac xxx.java` ⇒ `xxx.class`
  - ☞ `rmic xxx` ⇒ `xxx_Stub.class`  
`xxx_Skel.class`
3. Udostępnienie wygenerowanego kodu klas
  - ☞ wspólny system plików
  - ☞ kopia kodów klas w różnych systemach plików
  - ☞ udostępnianie kodu przez serwer www
4. Uruchomienie aplikacji
  - ☞ uruchomienie `rmiregistry` (name server)
  - ☞ uruchomienie serwera: utworzenie zdalnych obiektów i ich rejestracja w `rmiregistry`
  - ☞ uruchomienie klienta: zlokalizowanie zdalnych obiektów (odwołanie do `rmiregistry`) i wywołanie zdalnych metod

### Komunikacja między klientem a serwerem



### Interfejs `java.rmi.Remote`

Komunikacja pomiędzy serwerem (obiektem) a klientem jest określona przez definicję interfejsu pochodnego od interfejsu Remote. Klasa zdalnego obiektu musi implementować ten interfejs.

```
package compute;

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

public interface Compute extends Remote {
    Object executeTask(Task t) throws
        RemoteException;
}
```

### Zdalne udostępnianie obiektu (serwer)

#### Definicja klasy obiektu

```
package engine;

import java.rmi.*;
import java.rmi.server.*;
import compute.*;

public class ComputeEngine
    extends UnicastRemoteObject
    implements Compute
{
    public ComputeEngine() throws
        RemoteException {
        super();
    }

    public Object executeTask(Task t) {
        return t.execute();
    }
}
```

## Utworzenie i rejestracja zdalnego obiektu

```
public static void main(String[] args) {
    if (System.getSecurityManager() == null) {
        System.setSecurityManager(
            new RMISecurityManager());
    }
    String name = "//host:1099/Compute";
    try {
        Compute engine = new ComputeEngine();
        Naming.rebind(name, engine);
        System.out.println("ComputeEngine ok");
    } catch (Exception e) {
        System.err.println("ComputeEngine excep."
            + e.getMessage());
        e.printStackTrace();
    }
}
```

## Zdalne wywoływanie metod

```
package client;

import java.rmi.*;
import java.math.*;
import compute.*;

public class launchComp {
    public static void main(String args[]) {
        if (System.getSecurityManager() == null){
            System.setSecurityManager(
                new RMISecurityManager());
        }
        try {
            String name = "/" + args[0] + "/Compute";
            Compute comp =
                (Compute) Naming.lookup(name);
            CompArea task = new CompArea(...);
            BigDecimal arae = (BigDecimal)
                (comp.executeTask(task));
            System.out.println(arae);
        } catch (Exception e) {
            System.err.println("ComputePi excep."
                + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

## Przekazywanie parametrów

Jeżeli przekazujemy obiekt przez wartość, to klasa tego obiektu musi implementować interfejs `Serializable`.  
Jeśli przekazujemy obiekt przez referencja, to klasa tego obiektu musi implementować interfejs `Remote`.

```
package compute;

public interface Task extends
    java.io.Serializable {
    Object execute();
}
```

```
package client;

import compute.*;
import java.math.*;

public class CompArea implements Task {
    :
}
```

## Lokalizowanie zdalnych obiektów

- ☞ Rejestr — wiąże z nazwami i udostępnia zdalne obiekty
- ☞ Tworzenie rejestru — `rmiregistry [<port number>]`
- ☞ Klasa `LocateRegistry` — umożliwia tworzenie rejestru (obiekty klasy `Registry`)
- ☞ Klasa `Registry` — klasa obiektu-rejestru
- ☞ Klasa `Naming` — ułatwia korzystanie z rejestru, umożliwiając jego specyfikację w adresie URL

### Klasa LocateRegistry

```
static
Registry createRegistry(int port)
    throws RemoteException

static
Registry createRegistry(String host, int port)
    throws RemoteException

static
Registry getRegistry(int port)
    throws RemoteException

static
Registry getRegistry(String host, int port)
    throws RemoteException
```

### Klasa Registry

```
void bind(String name, Remote obj)
    throws RemoteException,
    AlreadyBoundException, AccessException

void rebind(String name, Remote obj)
    throws RemoteException, AccessException

void unbind(String name)
    throws RemoteException,
    NotBoundException, AccessException

String[] list()
    throws RemoteException, AccessException

Remote lookup(String name)
    throws RemoteException,
    NotBoundException, AccessException
```

### Klasa Naming

```
static
void bind(String urlName, Remote obj)
    throws RemoteException,
    AlreadyBoundException, AccessException,
    MalformedURLException, UnknownHostException

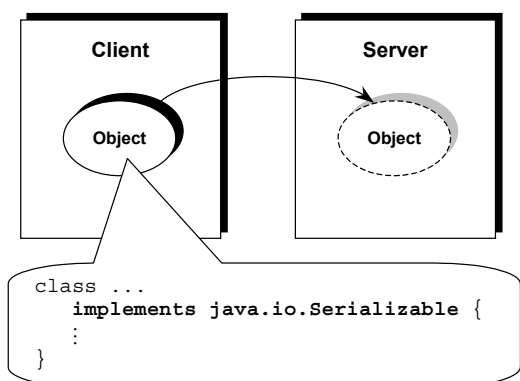
static
void rebind(String urlName, Remote obj)
    throws RemoteException, AccessException,
    MalformedURLException, UnknownHostException

static
void unbind(String urlName)
    throws RemoteException, NotBoundException,
    AccessException, MalformedURLException,
    UnknownHostException

static
String[] list()
    throws RemoteException, AccessException,
    MalformedURLException, UnknownHostException

static
Remote lookup(String urlName)
    throws RemoteException, NotBoundException,
    AccessException, MalformedURLException
```

### Przekazywanie parametrów przez wartość



## Przekazywanie parametrów przez referencję

