

ORPC

UDIS 2011/2012

ORPC (objektni RPC)

- ORPC tehnologija omogućava razvoj mrežno distribuiranih aplikacija koje su objektno strukturirane (aplikacija je kolekcija objekata).
- Servisi i resursi distribuiranih aplikacija su dohvatljivi kao objekti. Objekti imaju definirane metode, pa pozivanjem metode objekata distribuirana aplikacija pristupa servisima druge aplikacije.
- Objektni RPC je prirodna ekstenzija RPC na objektno-orijentiranje programiranje.

ORPC

- ORPC tehnologija se pojavljuje u 90-tima sa ciljem implementacije objektivne programske paradigme kojom se funkcionalnosti aplikacije enkapsuliraju unutar metoda u objektu.
- Poznata ORPC rješenja su:
 1. CORBA (*Common Object Request Broker Architecture*)(ver 1.1 - 1991., 2.0 - 1996., 3.0 -2002.)
 2. DCOM (*Distributed Component Object Model*)
 3. Java RMI (*Remote Method Invocation*)
 4. .NET Remoting

ORPC

- Tehnologija funkcionira slično kao i “neobjektni” RPC, sa IDL jezikom kojim se opisuje sučelje servera.
- Kompajliranjem se dobivaju klijentski i serverski *stub* u kojima se mapiraju objekti i na serverskoj i na klijentskoj strani.
- Međusloj rješava razlike u formatiranju postupkom *marshallinga/unmarshallinga*.

ORPC

- Načini adresiranja, vezivanja, obrade grešaka itd. su definirani međuslojem.
- Višestruko nasljeđivanje rješava se ovisno o odgovarajućem modelu nasljeđivanja određenog programskog jezika.

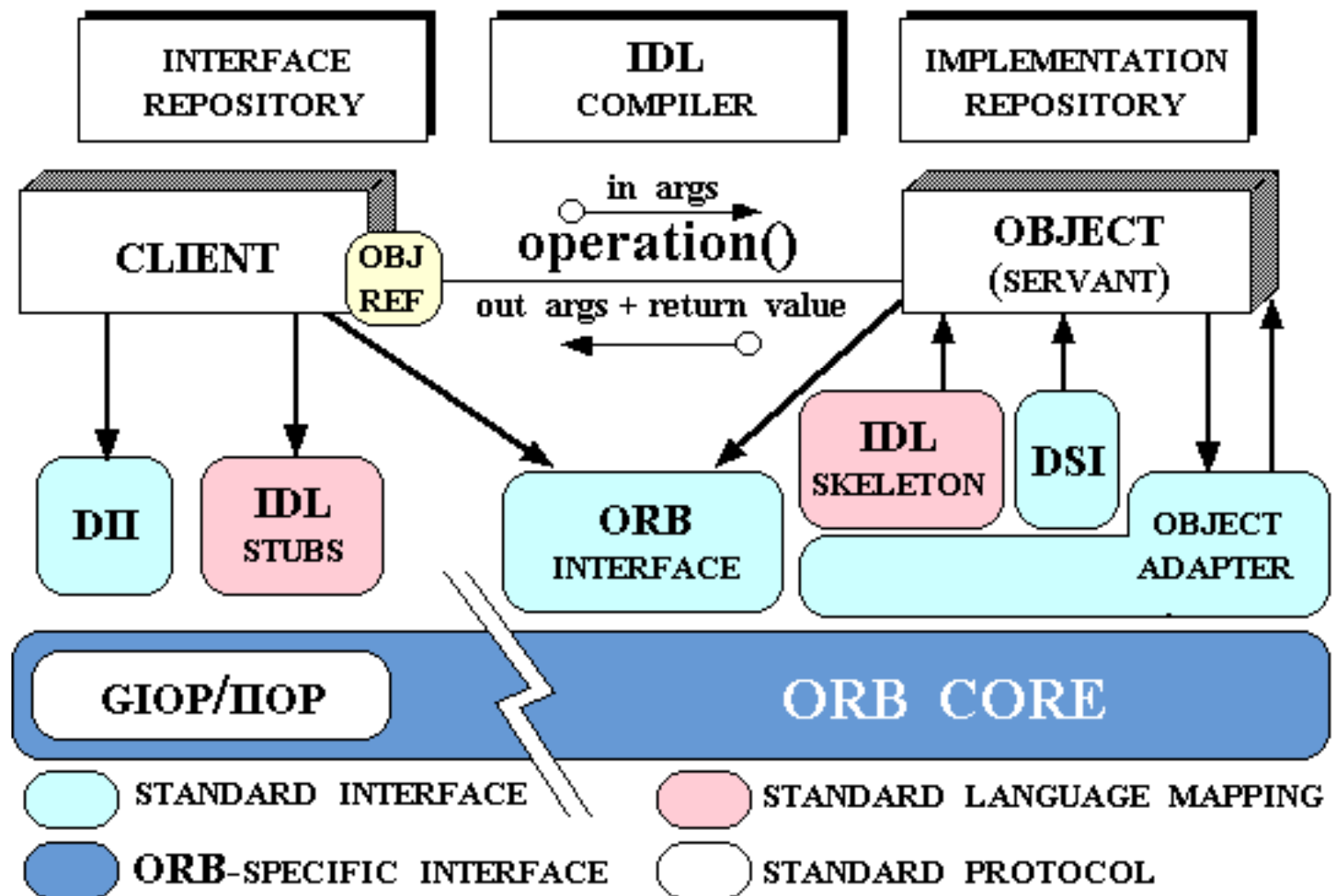
Corba

- 1989. grupa proizvođača i korisnika je osnovala *Object Management Group* (OMG), grupaciju koja je trebala unijeti reda u brzi i razjedinjeni razvoj objektne tehnologije.
- Ciljevi OMG-a bili su promoviranje objektne tehnologije i usmjeravanje razvoja osnivanjem “referentnog modela arhitekture upravljanja objekata” (*Object Management Architecture Reference Model*) prema kojem se temelje sve OMG specifikacije.
- Jedna od specifikacija je i CORBA.

Corba

- To je univerzalni, OO međusloj i arhitektura koja programerima omogućava pisanje objekata koji komuniciraju s drugim objektima bez da znaju gdje se objekti nalaze i kako su implementirani.
- Podržava izradu i integraciju objektno orijentiranih softverskih komponenti u heterogeno distribuiranim okruženjima.
- Ovo nije gotovi API za razvoj aplikacija, već samo specifikacija prema kojoj se može razviti vlastiti ORPC međusloj sa API-jem za razvoj aplikacija.

CORBA architektura



CORBA arhitektura

- IIOP (*Internet Inter-ORB Protocol*) je CORBA OMG komunikacijski protokol koji se oslanja na HTTP-a kao transportni protokol.
- Spremište sučelja (*Interface repository*) je izvršno spremište meta podataka registriranih IDL-definiranih sučelja koje sadrži imena metoda i tražene parametre. Sučelje se koristi za generiranje poziva udaljenih metoda.

CORBA arhitektura

- Implementacijsko spremište (*Implementation repository*) je izvršna baza podataka koja pohranjuje sve registrirane objekte nekog ORB-a (Object Request Broker), bilo da su aktivirani ili dostupni za aktivaciju nakon klijentskog zahtjeva.
- ORB je zadužen za pronalaženje objekta, poziv metode i vraćanje odgovora pozivatelju kada klijent pozove metodu na objektu.

CORBA arhitektura

- IDL stubovi i skeletoni omogućavaju pristup do IDL definiranih operacija na nekom objektu.
- Većina ORB funkcionalnosti ostvaruje se preko IDL stubova ili preko sučelja za dinamičke pozive (*dynamic invocation interface*), okosnica (*skeleton*) ili objektnih prilagodnika (*object adapter*).
- *Basic Object Adapter* predstavlja način na koji implementacije objekata dohvaćaju servise koje im omogućavaju ORB-ovi.

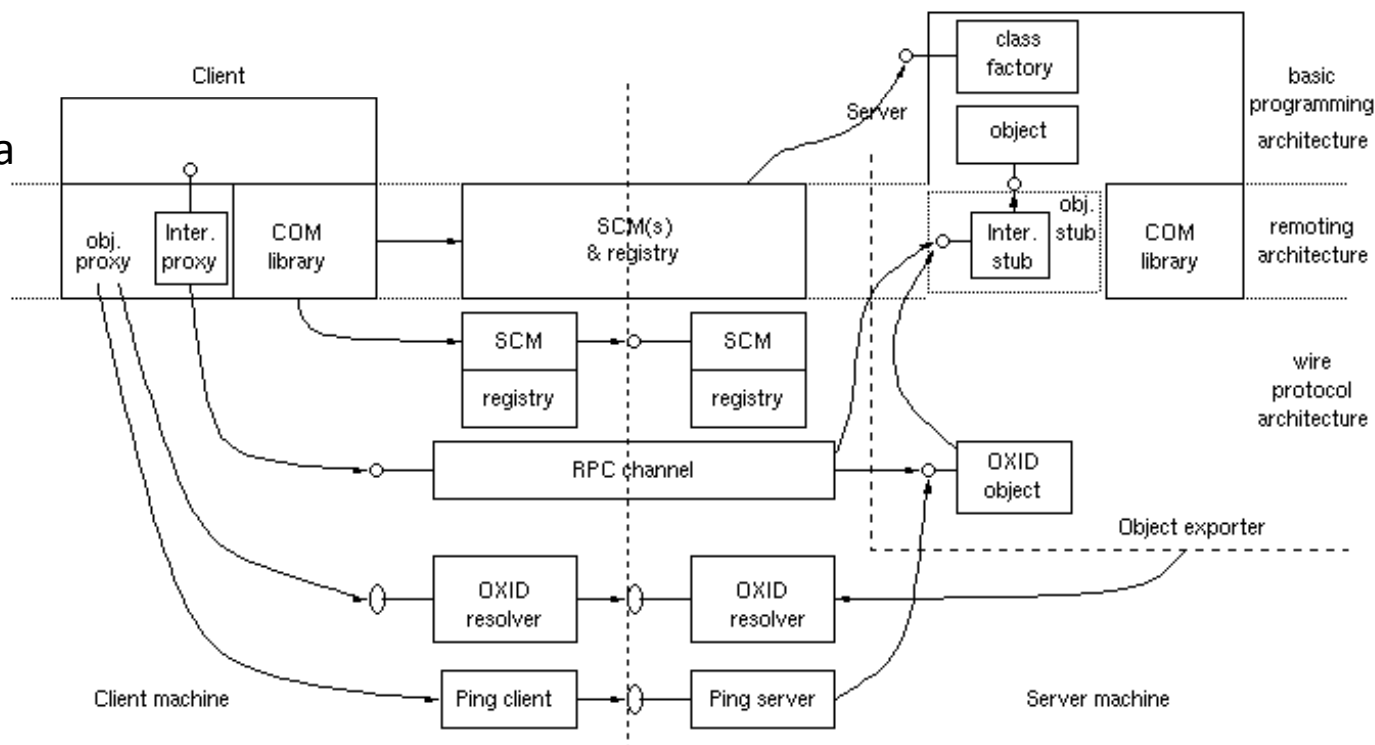
Microsoft COM

- Microsoftov objektni model distribuiranih komponenti pojavio se 1990. kada su predstavili svoje proizvode nazvane *Dynamic Data Exchange* (DDE) i *Object Linking and Embedding* (OLE).OLE2 nazvan Automatizirani OLE, uslijedio je 1993.
- Microsoft je model nazvao model objektnih komponenti (COM - *component object model*), promoviravši ga kao infrastrukturu opće namjene za izgradnju softverskih aplikacija sastavljenih od komponenti namijenjenih istom računalu.

Microsoft DCOM

- DCOM (*Distributed Common Object Model*) proširuje funkcionalnost COM-a na izgradnju softverskih aplikacija sastavljenih od mrežno distribuiranih komponenti ugradnjom mogućnosti poziva udaljenih metoda, sigurnosnim provjerama, skalabilnošću i lokacijskom transparentnošću.

DCOM
arhitektura



DCOM ↔ CORBA

DCOM IDL

```
// uuid and definition of IGrid1
[
    object,
    uuid(3CFDB283-CCC5-11D0-BA0B-00A0C90DF8BC),
    helpstring("IGrid1 Interface"),
    pointer_default(unique)
]
interface IGrid1 : IUnknown
{
    import "unknwn.idl";
    HRESULT get([in] SHORT n, [in] SHORT m, [out] LONG *value);
    HRESULT set([in] SHORT n, [in] SHORT m, [in] LONG value);
};

// uuid and definition of IGrid2
[
    object,
    uuid(3CFDB284-CCC5-11D0-BA0B-00A0C90DF8BC),
    helpstring("IGrid2 Interface"),
    pointer_default(unique)
]
interface IGrid2 : IUnknown
{
    import "unknwn.idl";
    HRESULT reset([in] LONG value);
};

// uuid and definition of type library
[
    uuid(3CFDB281-CCC5-11D0-BA0B-00A0C90DF8BC),
    version(1.0),
    helpstring("Grid 1.0 Type Library")
]
library GRIDLib
{
    importlib("stdole32.tlb");
    // uuid and definition of class
    [
        uuid(3CFDB287-CCC5-11D0-BA0B-00A0C90DF8BC),
        helpstring("Grid Class")
    ]
    // multiple interfaces
    coclass CGrid
    {
        [default] interface IGrid1;
        interface IGrid2;
    };
};
```

DCOM server main program

```
HANDLE hevtDone;

void main()
{
    // Event used to signal this main thread
    hevtDone = CreateEvent(NULL, FALSE, FALSE, NULL);
    hr = CoInitializeEx(NULL, COINIT_MULTITHREADED);
    CClassFactory* pcf = new CClassFactory;
    hr = CoRegisterClassObject(CLSID_CGrid, pcf,
        CLSCTX_SERVER, REGCLS_MULTIPLEUSE, &dwRegister);
    // Wait until the event is set by CGrid::~CGrid()
    WaitForSingleObject(hevtDone, INFINITE);
    CloseHandle(hevtDone);
    CoUninitialize();
}
```

CORBA server main program

```
int main()
{
    // create a grid object using the implementation class grid_i

    grid_i ourGrid(100,100);

    try {
        // tell Orbix that we have completed the server's initialization:
        CORBA::Orbix::impl_is_ready("grid");
    } catch (...) {
        cout << "Unexpected exception" << endl;
        exit(1);
    }
}
```

CORBA IDL

```
interface grid1
{
    long get(in short n, in short m);
    void set(in short n, in short m, in long value);
};

interface grid2
{
    void reset(in long value);
};

// multiple inheritance of interfaces
interface grid: grid1, grid2
{
};
```

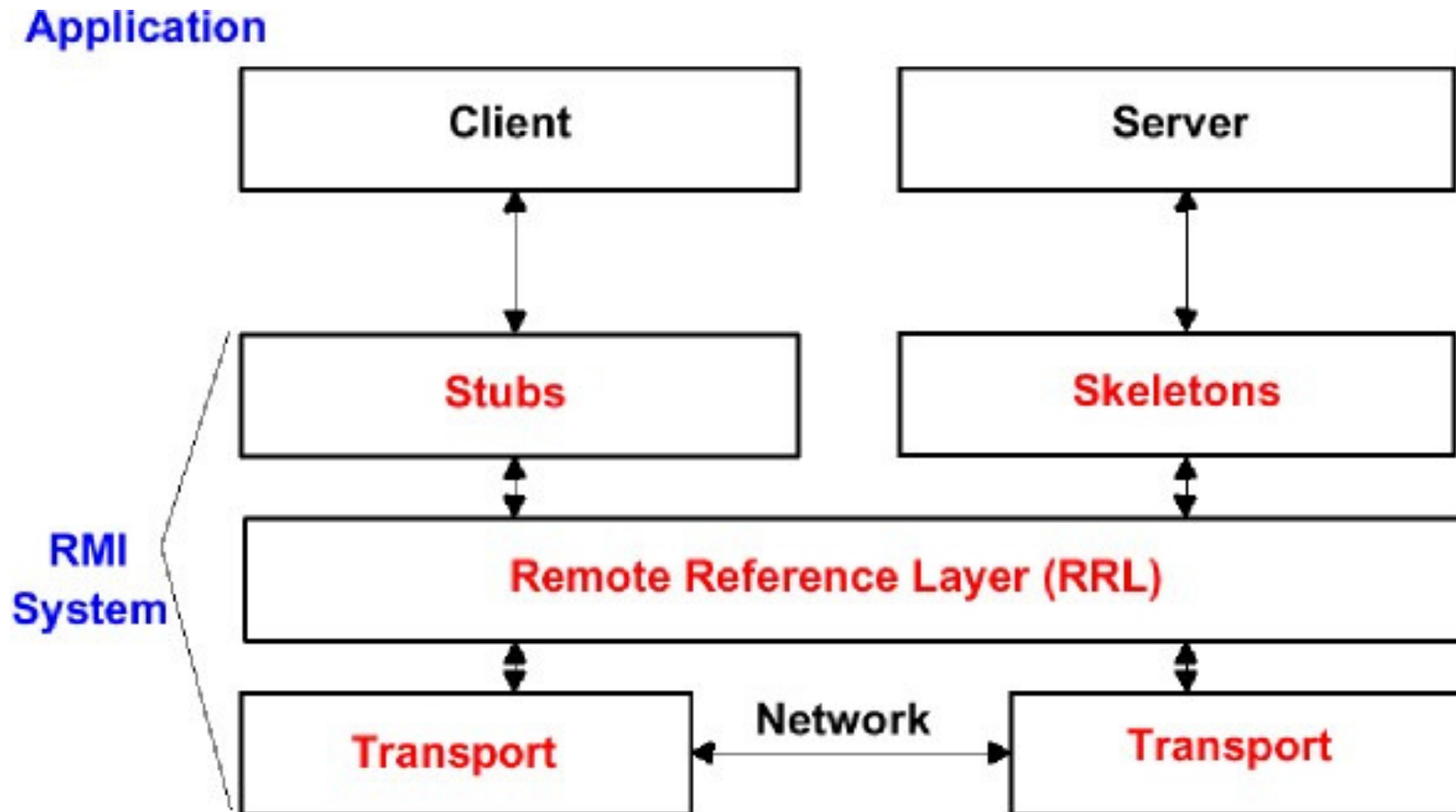
RMI

- Java RMI (*Remote Method Invocation*) je tehnologija objektnog RPC u Java programskom jeziku uvedena od JDK (Java Development Kit) verzije 1.1.
- Za razliku od CORBA-e, RMI je potpuno integriran sa Java jezikom i njenim okruženjem.
- Tako su CORBA sučelja opisana korištenjem arhitekturno neutralnih IDL-ova, a sučelja udaljenih Java objekata opisana su korištenjem klasičnog Java sučelja (interface).

RMI

- Kao i sve ostale tehnologije za razvoj distribuiranih aplikacija RMI ima mehanizam za adresiranje udaljenih objekata (URL sintaksa).
- Kao transportni protokol se koristi TCP/IP.
- Prijenos parametara (*marshalling/unmarshalling*) između distribuiranih procesa se radi postupkom serijalizacije/deserijalizacije. To je postupak pretvaranja objekata u stream bajtova (i obrnuto).

RMI architektura



Kreiranje RMI aplikacije

- Postupak kreiranja RMI aplikacije:
 1. Definiranje sučelje servera
 2. Implementacija sučelja
 3. Generiranje *stuba*
 4. Implementacija servera
 5. Implementacija klijenta

RMI definicija sučelja - primjer

```
import java.rmi.*;

public interface Matematika
    extends Remote
{
    public float kvadriraj(float )
        throws RemoteException;
}
```

- Sučelje treba public
- Treba proširivati (nasljeđivati) standardno java.rmi.Remote sučelje
- Metode trebaju podizati RemoteException

RMI implementacija sučelja - primjer

```
import java.rmi.*;
import java.rmi.server.*;
public class Matematika extends
    UnicastRemoteObject implements
    MatematikaInterface
{
    private float rezultat;

    public Matematika () throws
        RemoteException
    { }

    public float kvadriraj(float vrijednost)
        throws RemoteException
    {
        rezultat = vrijednost*vrijednost;
        return rezultat;
    }
}
```

- Klasa koja implementira definirano sučelje servera treba nasljediti klasu *java.rmi.server.UnicastRemoteObject* da bi se takvi objekti mogli koristiti kao udaljeni objekti.
- Klasa treba imati definiran konstruktor koji podiže *RemoteException*.
- Klasa treba implementirati metode definirane sučeljem koje se onda mogu pozivati od strane udaljenih klijenata. Klasa može imati i “svoje” metode koje se onda mogu pozivati samo lokalno.

RMI

- Kreiranje stuba se radi kompajliranjem implementiranog sučelja

1. `javac Hello.java`

- Nakon toga se poziva `rmic` kopajler koji generira *stub*

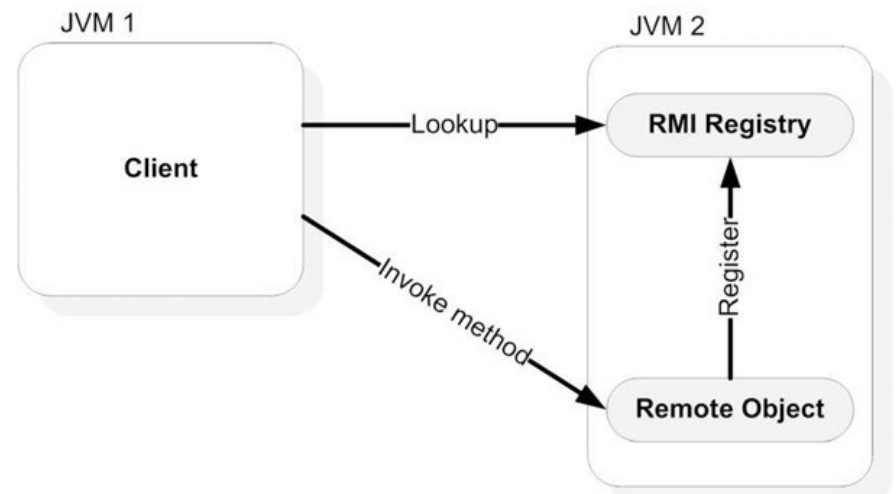
2. `rmic Hello`

Na Linux pokretanje procesa u backgroundu

- Pokretanje servera
- `rmiregistry &` - treba pokrenuti *Object Registry*
- `java HelloServer &`
- `java HelloClient`

RMI

- `rmiregistry` pokreće registar udaljenih objekata (RMI registry) na lokalnom računalu gdje se server izvodi. RMI registar omogućava klijentu koji pristupi računalu da pronađe željeni objekt.
- Ako nije specificiran port, za registar se po defaultu koristi port 1099.



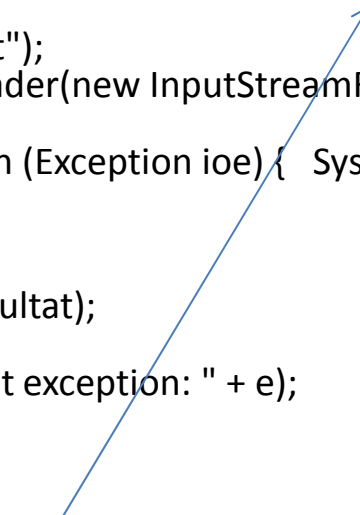
RMI server - primjer

```
import java.rmi.*;
class MatematikaServer
{
    public static void main (String[]
        argv) {
        try {
            Naming.rebind ("Matematika",
                new Matematika ());
            System.out.println ("Matematika
                Server is ready.");
        }
        catch (Exception e) {
            System.out.println ("Matematika
                Server failed: " + e);
        }
    }
};
```

- Naredba Naming.rebind kreira instancu servera prema korištenom sučelju (drugi parametar) i registrira ga u RMI registru sa navedenim imenom.
- Na taj se način povezuje referenca objekta sa imenom, kako bi klijent mogao pronaći objekt u registru koristeći navedeno ime.

RMI klijent - primjer

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.rmi.*;
class MatematikaClient
{
    public static void main (String[] argv) {
        try {
            MatematikaInterface klijent = (MatematikaInterface) Naming.lookup ("//localhost/Matematika");
            float upisi, rezultat;
            System.out.println ("upisite vrijednost");
            BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
            String pomocni = null;
            try { pomocni = br.readLine(); } catch (Exception ioe) { System.out.println("IO error !");
                System.exit(1); }
            upisi = Float.parseFloat(pomocni);
            rezultat = klijent.kvadriraj(upisi);
            System.out.println ("rezultat je " + rezultat);
        } catch (Exception e) {
            System.out.println ("MatematikaClient exception: " + e);
        }
    }
};
```



Spajanje klijenta na server