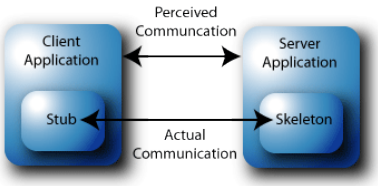
Software Ingeniaritza



4. Gaia: Inplementazioa

4.3 Konputazio banatua: Java RMI

A. Goñi, J. Ibáñez, J. Iturrioz, J.A. Vadillo

informatika fakultatea — facultad de informática

eman ta zabal zazu

Universidad del País Vasco — Euskal Herriko Unibertsitatea

Aurkibidea

RMI: Sarrera

RMI aplikazio baten garapena

- Urruneko interfazea definitu
- Urruneko interfazea inplementatu. Urruneko klasea
- Urruneko klasearen objektua sortu eta erregistratu
- Urruneko objektua bilatu eta exekutatu

RMI arkitektura

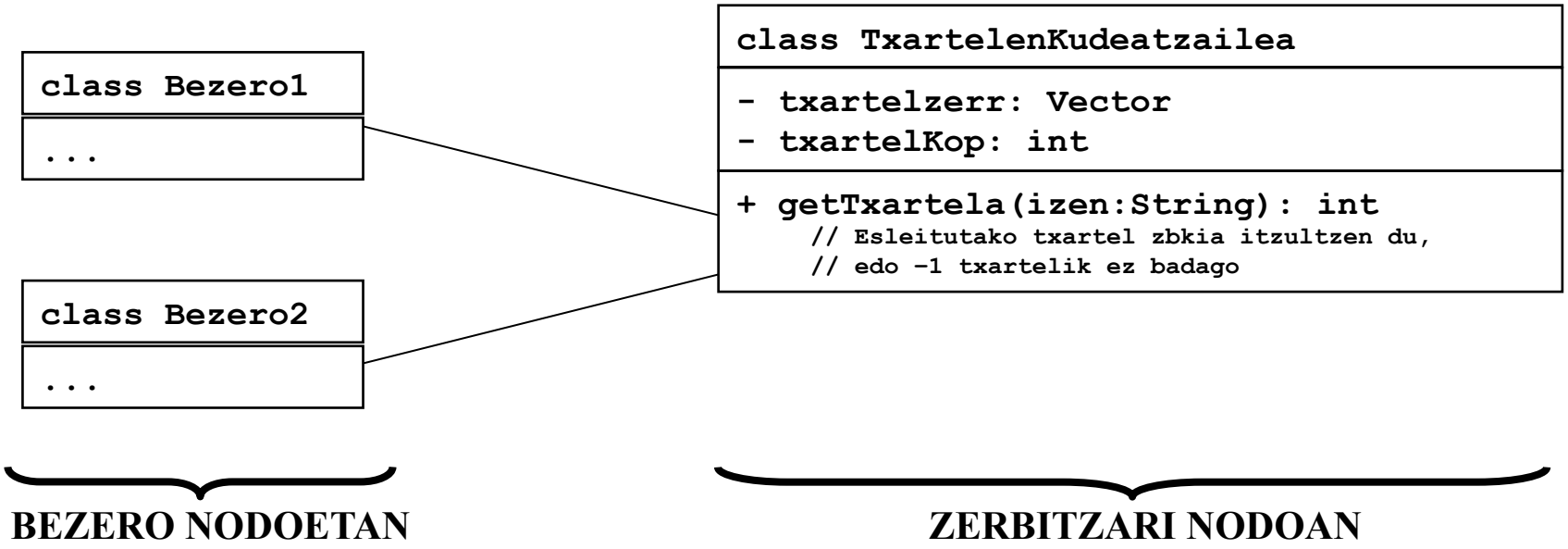
RMI: Sarrera

- RMI
  - API bat da,
  - java.rmi paketean aurkitzen diren klase eta metodo multzoa da.
    - aplikazio banatuak Javan garatzeko balio du, aplikazio ez banatuen sintaxi eta semantika berdinarekin.
- RPCren baliokidea.
- Badira beste estandar batzuk:
    - CORBA: zabalagoa da programazio lengoai ezberdinekin eraikitako zerbitzuak deitzeko ere balio du (eta ez soilik Java).

RMI: Remote Method Invocation

RPC: Remote Procedure Call
API : Application Programming Interface

RMI: Sarrera



Horrelako arkitektura batean ezin dugu bezero klase batean ondokoa egin:

```
TxartelenKudeatzailea k = new TxartelenKudeatzailea();
return k.getTxartela("Ibone Maia");
```

k urruneko Java makina birtualeko klase bateko objektua delako.

Aurkibidea

RMI: Sarrera

RMI aplikazio baten garapena

- Urruneko interfazea definitu
- Urruneko interfazea inplementatu
- Urruneko klasearen objektua erregistratu
- Urruneko objektua bilatu eta exekutatu

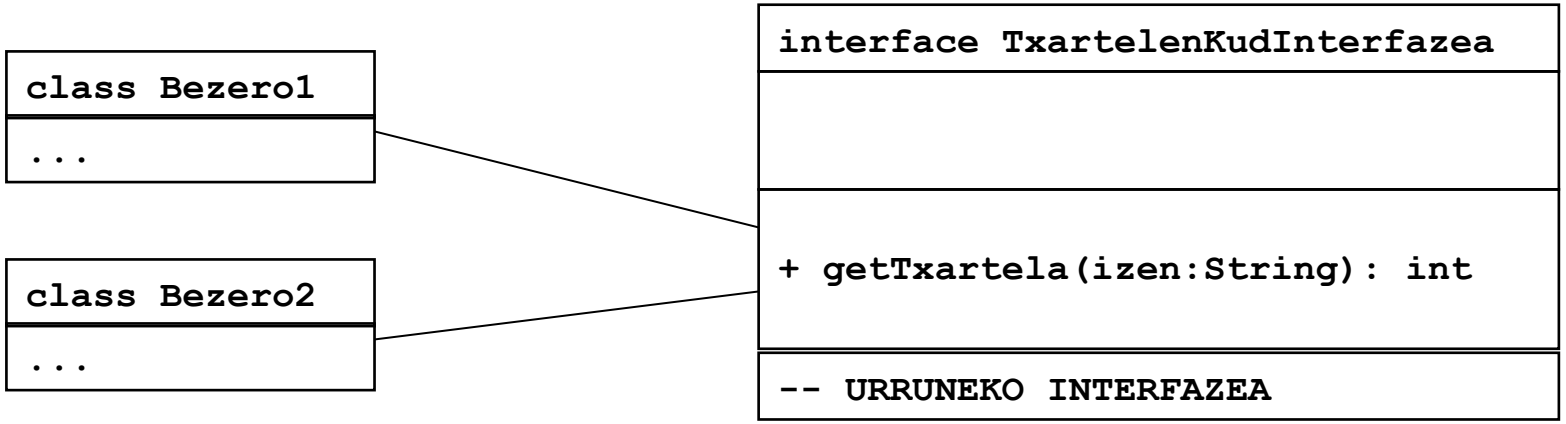
RMI arkitektura

RMI: Sarrera

Nola eraiki Bezero/Zerbitzari aplikazio bat RMI erabiliz, bezeroak urruneko zerbitzu (urruneko klase) bat atzitu dadin?

1. Urruneko interfaze bat definitu.
2. Urruneko interfaze hori inplementatu (Urruneko klasea).
3. Urruneko klasearen objektu bat sortu eta erregistratu.
4. Urruneko objektua bilatu eta exekutatu.

1 Urruneko interfazea definitu



RMI-k urruneko objektua deitzeko aukera ematen du.

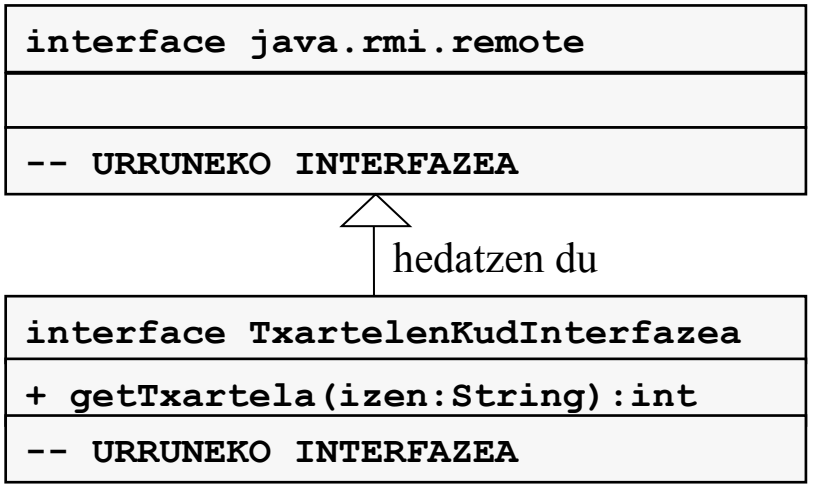
Horretarako urruneko interfaze bat definitu behar da.

Horrela bezero klase batek ondokoa egin dezake:

```
TxartelenKudInterfazea k ;
// Urruneko objektuaren helbidea jasotzeko
// eta k objektuan kodea uzteko
return k.getTxartela(“Ibone Maia”);
```

1 Urruneko interfazea definitu

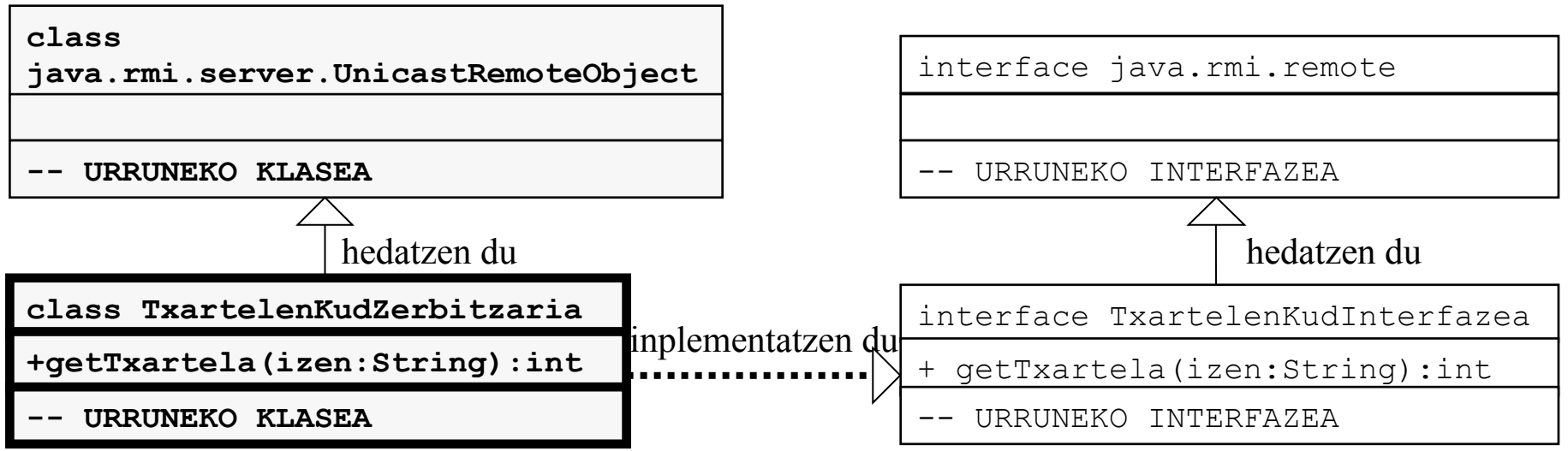
```
// TxartelenKudInterfazea.java
import java.rmi.*;
public interface TxartelenKudInterfazea extends Remote {
  public int getTxartela (String izen) throws RemoteException;
}
```



Urruneko interfazeak:

- **`java.rmi.Remote`** hedatzen du,
- bertan definitutako metodo guztiak **`java.rmi.RemoteException`** salbuespena altxa dezaten erazagutu behar da.

2 Urruneko interfazea inplementatu



Urruneko zerbitzari klaseak:

- **`java.rmi.server.UnicastRemoteObject`** hedatzen du,
- urruneko interfazearen metodoak inplementatzen ditu,
- bere klaseko objektu bat sortu eta izen batekin erregistratuko du (bezero klaseetako objektuek atzitu ahal izateko, eta bere metodoak urrunetik egikaritu ahal izateko).

2 Urruneko interfazea inplementatu

```
// TxartelenKudZerbitzaria.java
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;
import java.util.*;
public class TxartelenKudZerbitzaria
             extends UnicastRemoteObject
             implements TxartelenKudInterfazea {
  private Vector txartelZerr = new Vector();
  private static int txartelMax = 50;

  public TxartelenKudZerbitzaria() throws RemoteException {
  }
  public int getTxartela(String izen) throws RemoteException {
    //negozio  logika inplementatu
    return zenb;
  }
  ...
```

3 Urruneko klasearen objektu bat sortu eta erregistratu

- Erregistro bat sortzen da zerbitzarian (klase bereko main() metodoan edo beste klase desberdin batean)

```
java.rmi.registry.LocateRegistry.createRegistry(9999);
```

- Klase urruneko objektu bat sortzen da

```
TxartelenKudZerbitzaria
        zerbitzariObj =new TxartelenKudZerbitzaria();
```

- Izen bat definitzen da zerbitzurentzako

```
String zerbitzua = "//localhost/TxartelenKudeatzailea";
```

- Objektua erregistratzen da izen horrekin

```
Naming.rebind(zerbitzua,zerbitzariObj);
```

3 Urruneko klasearen objektu bat sortu eta erregistratu

```
public static void main(String[] args){
//Java policy falta da
 try {
   java.rmi.registry.LocateRegistry.createRegistry(1099);
 } catch (Exception e) {“Rmiregistry sortuta zegoen";}
 try {
   TxartelenKudZerbitzaria zerbitzariObj =new TxartelenKudZerbitzaria();

   String zerbitzua = “//localhost/TxartelenKudeatzailea”;
     //          “//localhost:Portuzenb/zerbitzuarenizena”;
  // Urruneko zerbitzua erregistratu
  Naming.rebind(zerbitzua,zerbitzariObj);
 }
 catch (Exception e){
   System.out.println(“Errorea zerbitzaria jaurtitzean”);
 }
}
```

3 Urruneko klasearen objektu bat sortu eta erregistratu

java.rmi.registry.LocateRegistry.createRegistry(p)

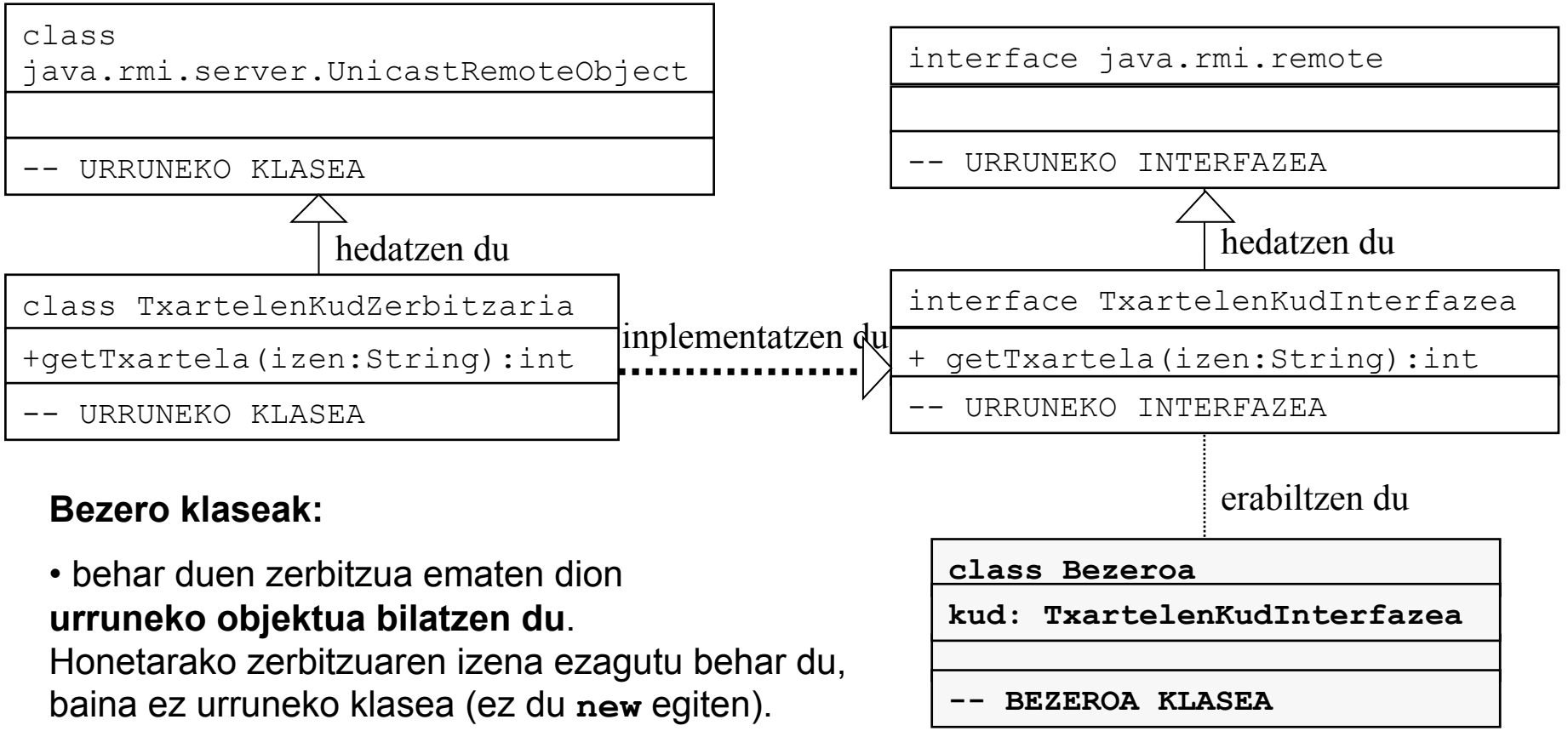
Agindu honek rmiregistry prozesua sortzen du p portuan.
Jaurtitako rmiregistry ez da bukatzen RMI zerbitzaria bukatzen bada ere.
Salbuespen bat goratzen du portua okupatua badago.

```
try { java.rmi.registry.LocateRegistry.createRegistry(1099);
} catch (Exception e)
{System.out.println("Rmiregistry martxan dago"+e.toString());}
```

RMI zerbitzaria jaurtitzen duen kodea eta salbuespena kontrolatzen du.
RMI zerbitzaria aldi bat baino gehiago exekutatzen bada.

rmiregistry gelditzeko:UnicastRemoteObject.*unexportObject*(registry, **true**);

4. Urruneko objetua bilatu eta exekutatu



Bezero klaseak:

• behar duen zerbitzua ematen dion **urruneko objektua bilatzen du**.
Honetarako zerbitzuaren izena ezagutu behar du, baina ez urruneko klasea (ez du **`new`** egiten).

• objektu horri eskatzen dio **urruneko metodo bat egikari dezan** (urruneko interfazean definitua).

4. Urruneko objetua bilatu eta exekutatu

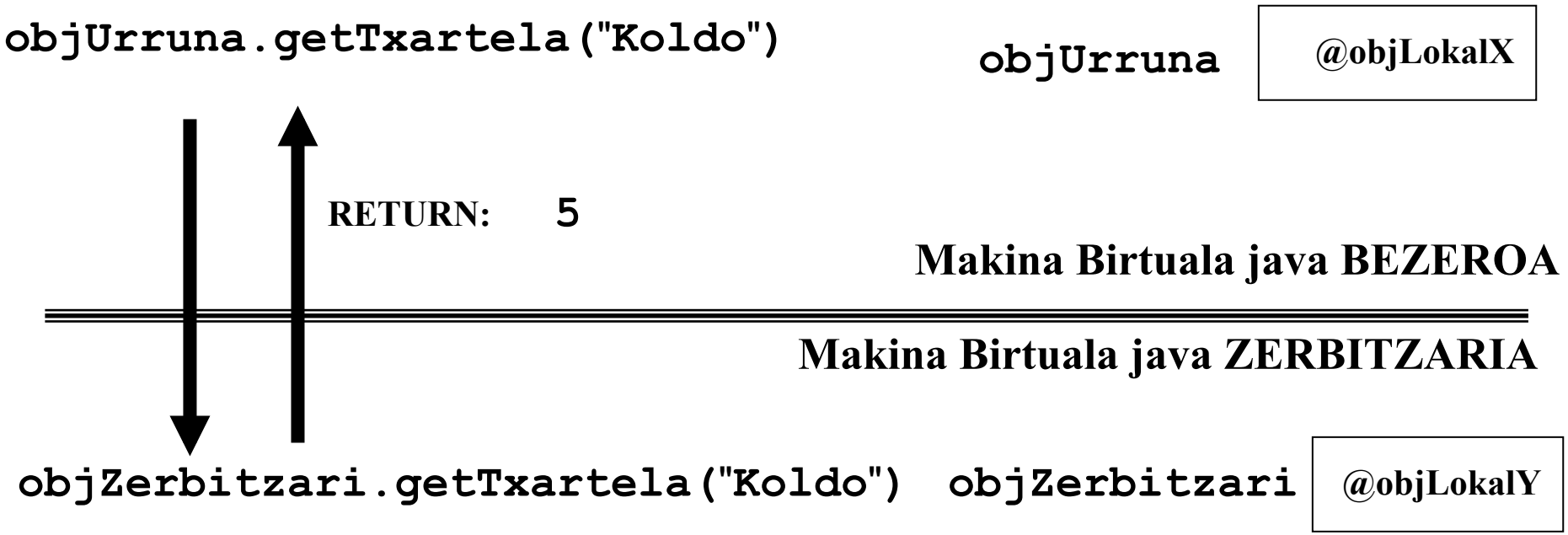
```
import java.rmi.*;
public class Bezeroa {
  public static void main(String[] args){
     System.setSecurityManager(new RMISecurityManager());
     TxartelenKudInterfazea urrunekoObj;
     String zerbIzena = "rmi://localhost/TxartelenKudeatzailea";
        //Makina berdinean
        // "rmi://IP_Helbidea:PortuZenb/ZerbitzuarenIzena"; //Beste makina batean
        // "rmi://sisx02.si.ehu.es/TxartelenKudeatzailea";  //Adibidez sisx02-an
     try {
     urrunekoObj =
         (TxartelenKudInterfazea)Naming.lookup(zerbIzena);

     int tx = urrunekoObj.getTxartela(args[0]);
     if (tx==-1) System.out.println("Bukatu dira txartelak");
     else System.out.println("Lortu da txartela: "+tx);
      }
      catch (Exception e){
     System.out.println("Errorea...");
      }
  }
}
```

Aurkibidea

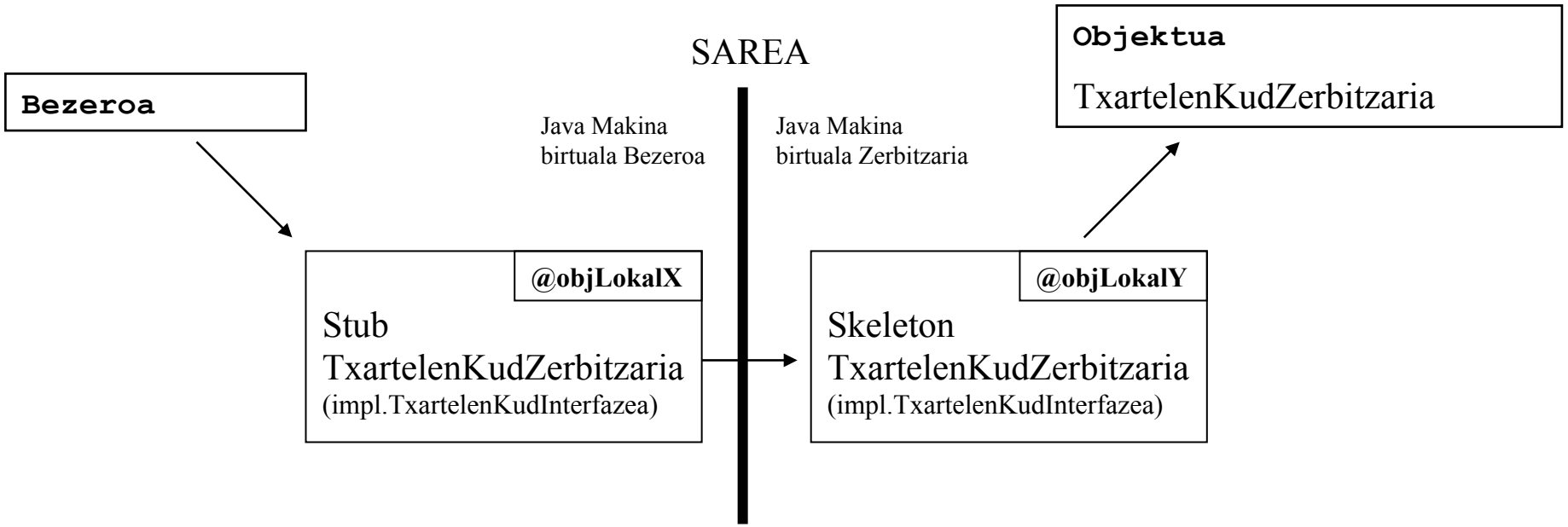
- RMI: Sarrera
- RMI aplikazio baten garapena
  - Urruneko interfazea definitu
  - Urruneko interfazea inplementatu. Urruneko klasea
  - Urruneko klasearen objektua sortu eta erregistratu
  - Urruneko objektua bilatu eta exekutatu
- RMI arkitektura

RMI Arkitektura



Lortu nahi dugu **`@objLokalX-ri`** eskatzen zaiona, **`@objLokalY`** exekutatu dadin beste Java makina birtual batean.

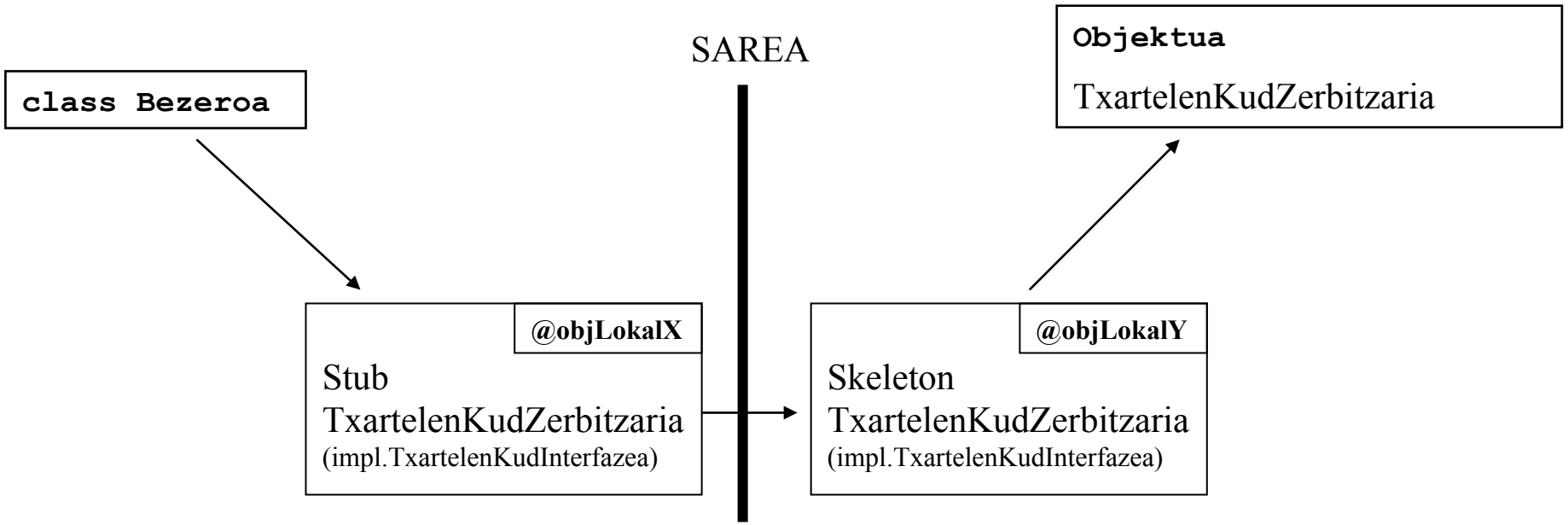
RMI arkitektura



1. Stub TxartelenKudZerbitzaria: Zerbitzaile klasearean ordezkaria bezero klasean.
2. Skeleton TxartelKudZerbitzaria: Zerbitzaile klasearen sareko kudeaketaren ordezkaria.

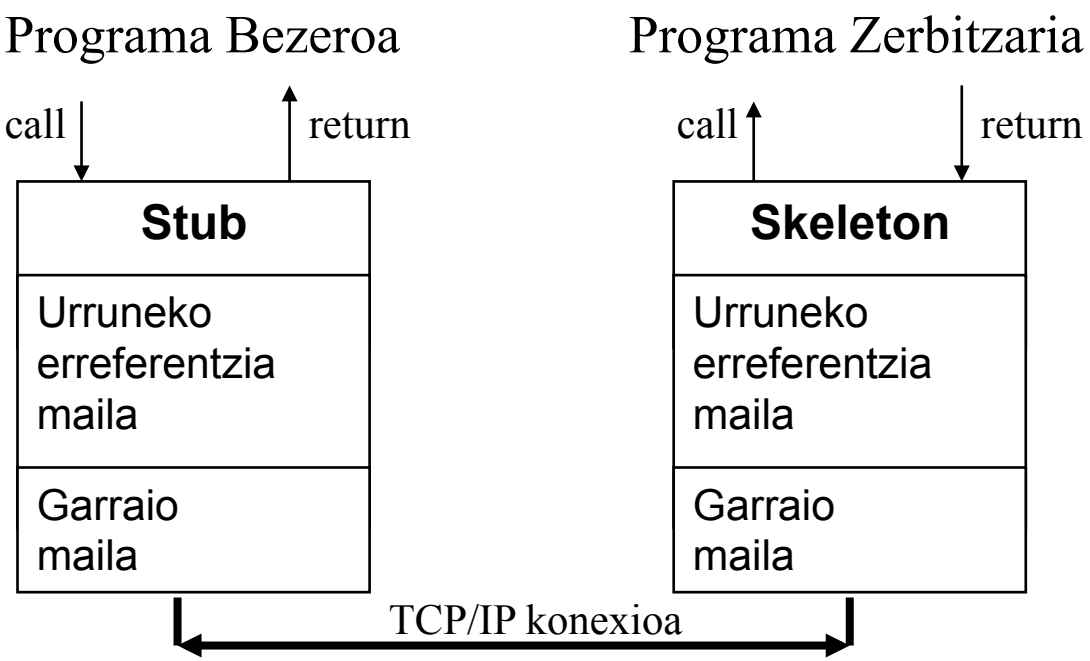
OHAR: Stub-ak, Skeleton-ak eta urruneko objektuak, interfaze berdina inplementatzen dute.

RMI arkitektura



1. Stub objektua objektu zerbitzariarekin (**`Skeleton)`** konektatu behar da. Skelotan-a berriz, urruneko objektuarekin konektatuko da (hau da, TxartelKudZerbitzaria objektuarekin).
2. Metodoetako parametroen balioak bezerotik zerbitzarira pasa behar dira.
3. Metodoetako emaitzak zerbitzaritik bezerora pasa behar dira.

RMI arkitektura



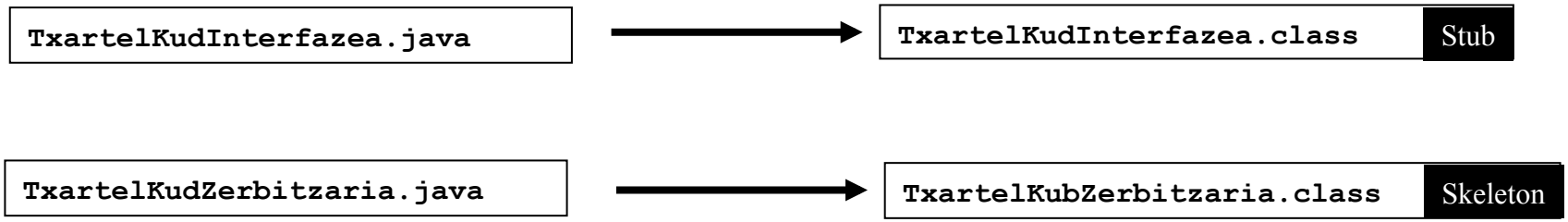
Stub eta Skeleton objektuak
konexioa egiteaz eta
parametroak eta emaitzak pasatzeaz
arduratzen dira.

Parametroen serializazioa

- Stub eta skeleton-ek bidalitako objektuak (parametroak eta emaitzak) **haien balioen kopia bat eginez** pasatzen dira (eta haiengan dauden objektuenak errekurtsiboki).
  - Ez da urruneko objektu bati erreferentziarik egiten.
- Horretarako Javaren serializazio mekanismoak erabiltzen dira.
  - **objektu hoien klaseak `Serializable` interfazea inplementatu behar dute.**

RMI Arkitektura

STUB eta SKELETON-a urruneko interfaze eta urruneko klasera asoziatuta daude



Konpilatzen direnean *Stub eta Skeleton kodea gehitzen zaie RMI definitutako klaseei* (jdk6.0)

TxartelKudInterfazea.class, bezeroarentzat eskuragarri geratu behar da.

rmiregistry – rebind (Zerbitzariak)

java.rmi.registry.LocateRegistry.createRegistry(p)
TxartelenKudZerbitzaria zerbitzariObj =new TxartelenKudZerbitzaria();

String zerbitzua = "//localhost/nireTxartelKud";

// Urruneko zerbitzua erregistratu
Naming.rebind(zerbitzua,zerbitzariObj);

IZENA	ZERBITZARI OBJKETUA	ZERBITZARI OBJEKTUA SKELETON
nireTxartelKud	@zerbitzariObj	@zerbitzariObj-Skeleton
...	...	...

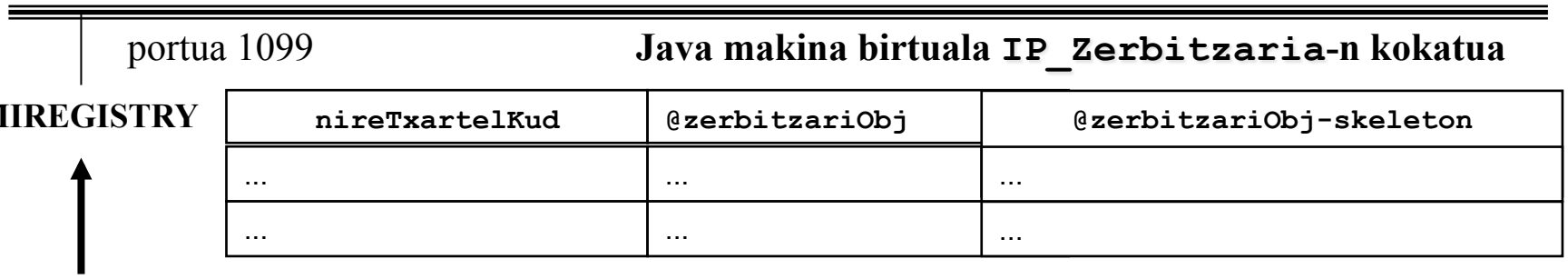
Rmiregistry logikoki Hash Taula egitura bat da kudeatzen du: Izen bakoitzentzako, objektu urruneko errefentzia bat gordetzen du (eta bere Skeleton objektua)

OHAR: Zerbitzua erregistratzen denean, Skeleton objektu bat sortzen da, bezeroen deialdiak entzuteko.

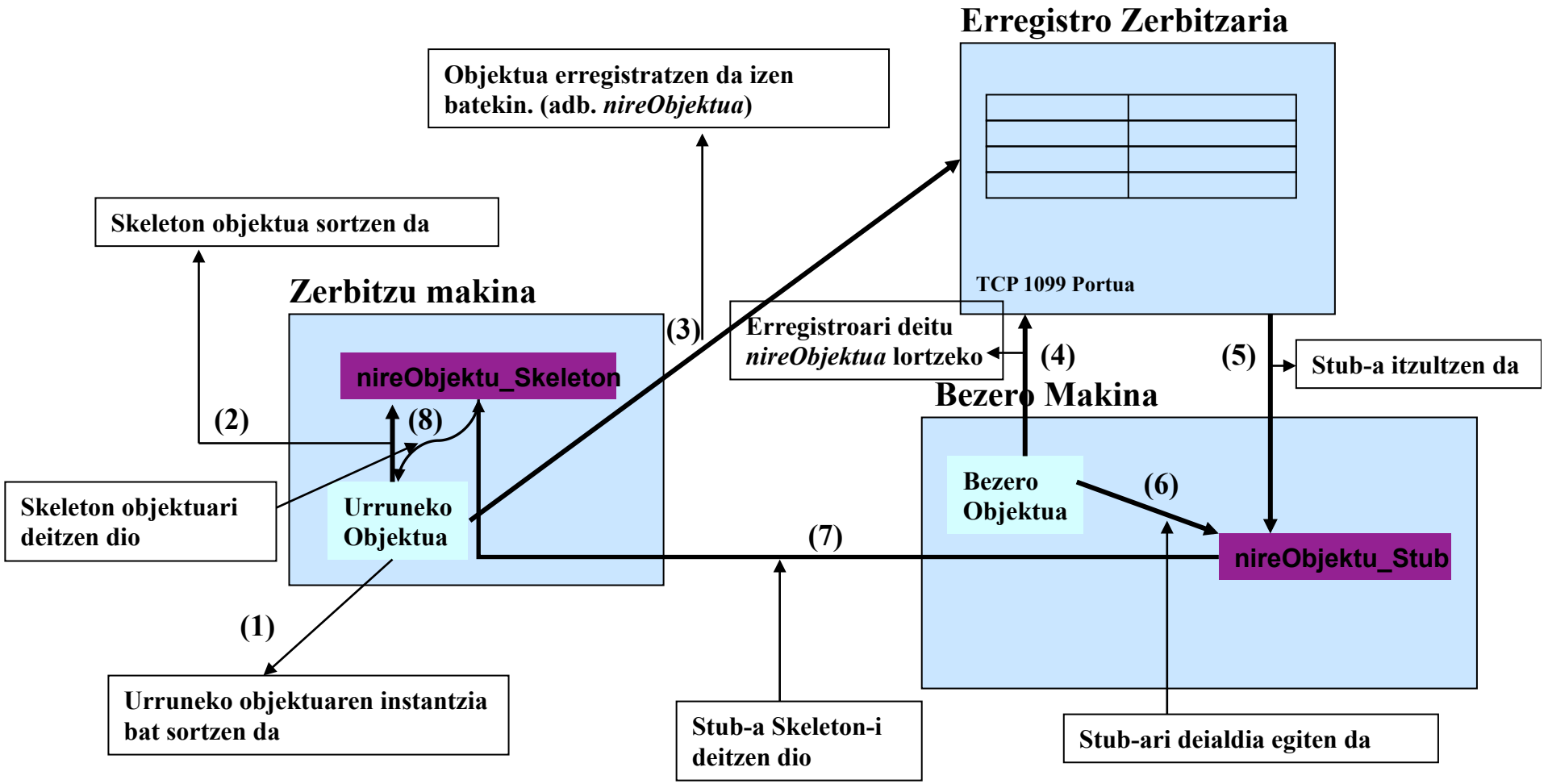
rmiregistry – lookup (Bezeroak)

```
TxartelenKudInterfazea urrunekoObj =
 (TxartelenKudInterfazea)Naming.lookup("rmi://IP_Zerbi/nireTxartelKud");
```

- lookup metodoak, urruneko objektuaren bezero ordezkari baten objektu bat itzultzen du (Stub objektu bat).
- Objektu hori, urruneko objektuaren sare ordezkariarekin konektatuko da (bere Skeleton objektua)



Komunikazio Eskema RMI arkitekturan



java.security.policy

- RMI-ko eragiketa batzuk, segurtasun politika kudeatzaile bat egotea eskatzen dute. Adib:
  - RMI-k klase serializable bat beste makina batetik kargatu ahal izango du soilik baimena ematen duen segurtasun kudeatzaile bat badago.
  - RMI-rako segurtasun kudeatzaile lehenetsi bat ezarri daiteke

    System.setSecurityManager(new RMISecurityManager());

- RMI-rako segurtasun kudeatzaile lehenetsiak oso politika murriztaile bat ezartzen du.
  - Soilik CLASSPATH lokaleko STUBak egikaritu daitezke.

java.security.policy

- Segurtasun kudeatzailea aldatu daiteke, Java makina birtualari segurtasun politikarako beste fitxategi bat adieraziz:

  System.setProperty("java.security.policy", “c:\\Proiektua\java.policy”);

- **`java.policy`** fitxategiaren edukia:

```
grant {
   permission java.security.AllPermission;
};
```