

# Distributed Applications

## Distributed Applications (I)

- Exchangeable components
- Independent maintenance
- Inherent need for distribution

Separation of interface / implementation

Non-distributed application      Distributed application

2

## Distributed Applications (II)

- The application code is distributed over several programs that are executed independently

Non-distributed application      Distributed application

3

## Distributed Applications (III)

Non-distributed application      Distributed application

4

## Communication between Processes

- Sockets: Client and Server communicate via i/o bytes.
- Connection to DBs (JDBC, ODBC): Client sends SQL Queries to a database server.
- Remote Procedure Call: Client sends Function calls to a server.
- Distributed objects (RMI, CORBA): Client sends Method calls and Object sending to a server.

5

# Distributed Objects

## OOP in Distributed Applications

Implementation

- Data structure
- Methods' code
- Object location**
- Class location (RMI)**
- Language (CORBA)**

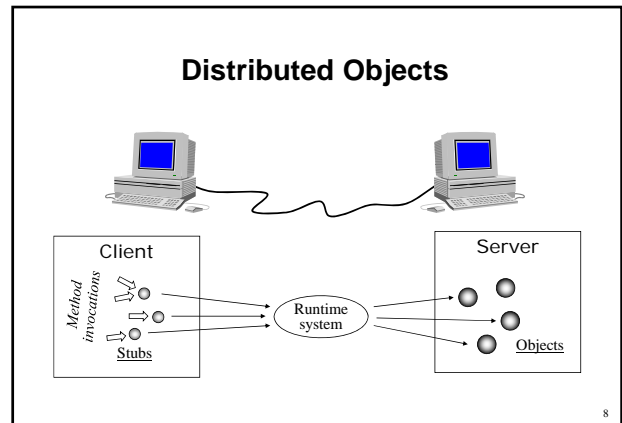
Interface

Method declaration

- Modularity
- Interface for the client
- Implementation on the server
- Transparency

Standards: RMI, CORBA, DCOM

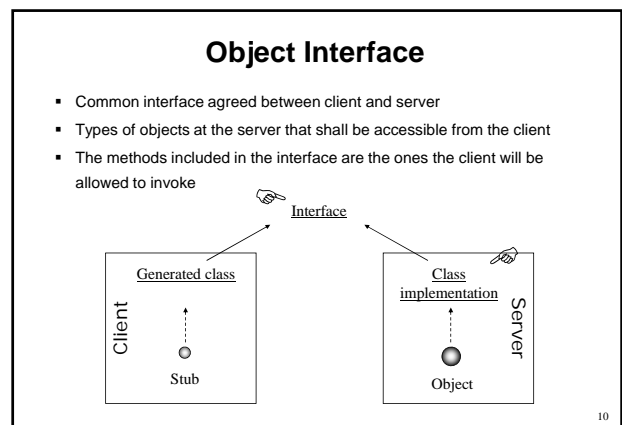
7



## Main Elements

- Remote interfaces
  - Agreed interface between server and client
  - Methods the client may invoke
- Object implementation
  - Implementation of interface methods
  - Objects are created at the server
- Stubs
  - Act as reference to remote objects from the client
  - Transfer to the server the calls from the client
- Runtime system
  - Mediator between stubs and remote objects
- Access to objects (obtaining stubs)
  - Naming service
  - Remote methods that return or send objects

9



## Interface: Example (RMI)

```

public interface RemoteAccount extends Remote {
    public long showBalance ()
        throws RemoteException;
    public void deposit (long amount)
        throws RemoteException;
    public void withdrawal (long amount)
        throws RemoteException;
};
    
```

11

## Objects and stubs

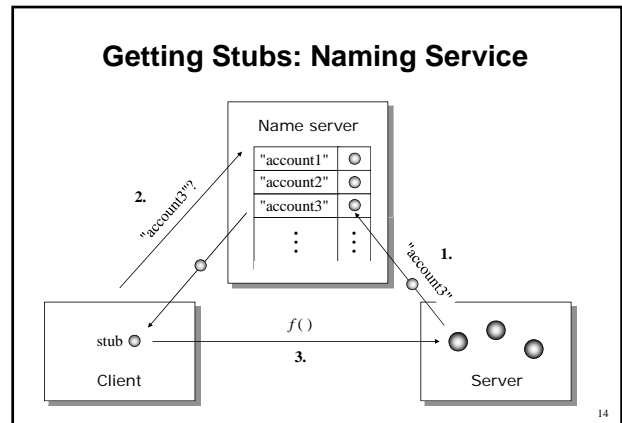
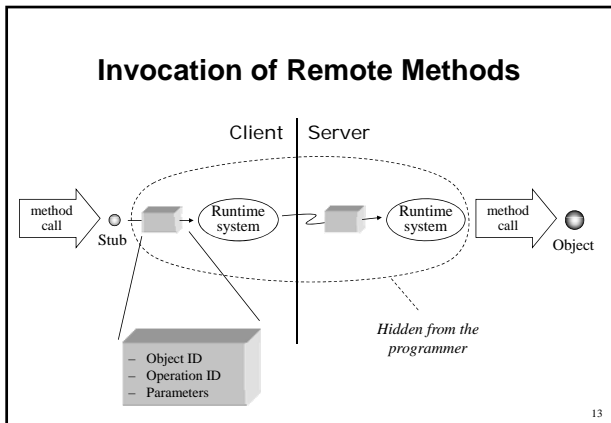
**Object implementation**

- Is part of the server program
- Must comply with the interface, may extend it
- The methods' implementation is executed on the server

**Stubs**

- Objects that reside on the client
- Represent remote objects (host + port + object ID)
- Their class is automatically generated from the interface
- They implement the interface ⇒ the methods can be called on them

12

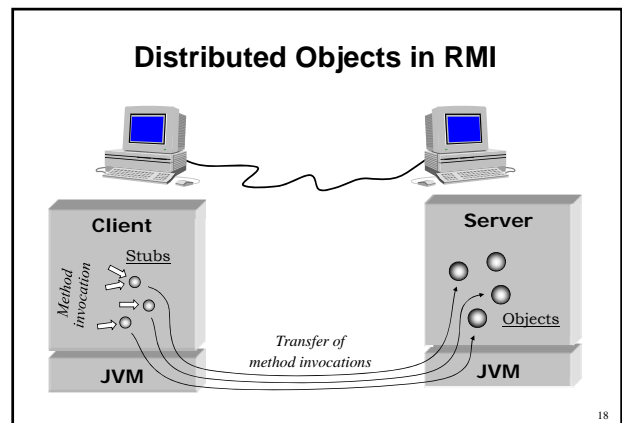


- ### Development Platforms for Distributed Objects
- Interface definition language
    - May be different from implementation language
  - Interface compiler
    - Generates classes for stubs
  - Runtime system
    - Mediator between stubs and object implementations
  - Naming service
    - Locating objects by name from the client
  - RMI, CORBA, DCOM
- 15

- ### Distributed Objects Standards
- #### Common Object Request Broker Architecture (CORBA)
- Created by the Object Management Group (OMG) → consensus
  - Integration of programs in different languages → heterogeneity
  - Interoperability (IIOP)
  - Additional integrated services
  - CORBA 1.0: 1992, CORBA 2.0: 1997, CORBA 2.6: dic 2001
  - Java 1.2 / 1.4 incorporate CORBA 2.0 / 2.3, named "Java IDL"
- #### Remote Method Invocation (RMI)
- Integrated in the Java language
  - Ease of programming, portability of implementations
  - Object pass by value
  - Dynamic class load
- 16

## Distributed Objects

### Remote Method Invocation (RMI)



### Building an Application in RMI

- Interface
- Implementation
- Server
- Client

Example

RemoteAccount.java  
 Account.java  
 Server.java  
 Client.java

19

### Interface and implementation

#### Interfaces

- Java interfaces that derive from the `java.rmi.Remote` interface
- All the methods must declare a `java.rmi.RemoteException`
- Arguments that the methods may have:
  - Primitive data types
  - Stubs and remote objects
  - Serializable local objects (which implement the `java.io.Serializable` class)

#### Implementation

- Subclass of `java.rmi.server.UnicastRemoteObject` that implements the remote interface
- Implement all the methods of the interface

20

### Define the interface

RemoteAccount.java

```

public interface RemoteAccount extends Remote {
    public long showBalance ()
        throws RemoteException;
    public void deposit (long amount)
        throws RemoteException;
    public void withdrawal (long amount)
        throws RemoteException;
    public void transfer (RemoteAccount account,
        long amount)
        throws RemoteException;
}
    
```

21

### Implement the Interface

Account.java

```

class Account extends UnicastRemoteObject
    implements RemoteAccount {
    private long balance;
    private long number;
    private static long numAccounts;
    Account () throws RemoteException {
        number = ++numAccounts;
    }
    public long showBalance ()
        throws RemoteException {
        return balance;
    }
}
    
```

22

```

public void deposit (long amount)
    throws RemoteException {
    balance += amount;
}
public void withdrawal (long amount)
    throws RemoteException {
    balance -= amount;
}
public void transfer (RemoteAccount account,
    long amount)
    throws RemoteException {
    withdrawal (amount);
    account.deposit (amount);
}
    
```

23

### Server Program and Client Program

- Server program
  - Creates instances of remote classes and registers them on the name server
- Client program
  - Declares objects of the remote interface and gets stubs from the name server
  - Invokes methods on the stubs
- Naming service
  - The naming server program: `rmiregistry`
  - The `java.rmi.Naming` class
 

<code>Naming.rebind (String, Remote)</code>	}	<i>Asignación de nombres</i>
<code>Naming.unbind (Remote)</code>	}	
<code>Naming.lookup (String) → Remote</code>	}	<i>Localización de objetos</i>

24

### Server Program

```
Server.java
public class Server {
    public static void main (String args[])
        throws Exception {
        Account c1 = new Account ();
        Naming.rebind ("account1", c1);
        Account c2 = new Account ();
        Naming.rebind ("account2", c2);
    }
}
```

25

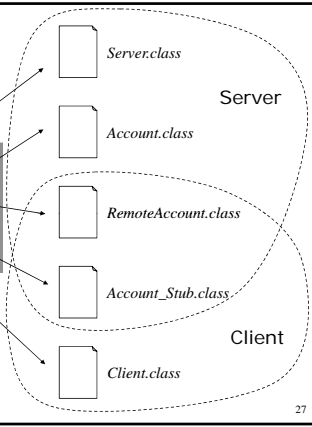
### Client Program

```
Client.java
public class Client {
    public static void main (String args[])
        throws Exception {
        RemoteAccount c1 = (RemoteAccount) Naming.lookup (
            "//example.uam.es/account1");
        RemoteAccount c2 = (RemoteAccount) Naming.lookup (
            "//example.uam.es/account2");
        c1.deposit (1000);
        c1.transfer (c2, 400);
        System.out.println ("Account 1 balance: "
            + c1.showBalance ())
            + "\nAccount 2 balance: "
            + c2.showBalance ());
    }
}
```

26

### Compilation

```
> javac Server.java
> javac Account.java
> javac RemoteAccount.java
> rmic Account
> javac Client.java
```

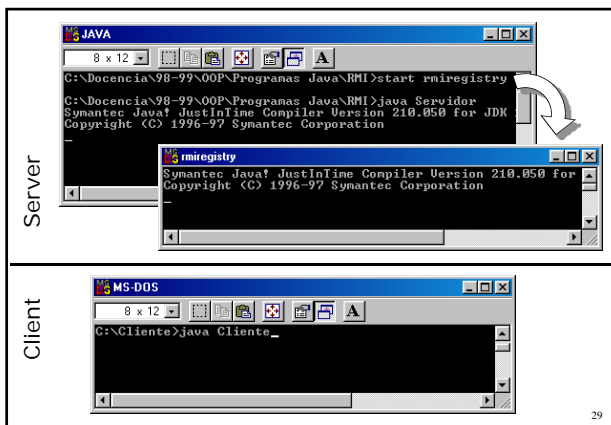


27

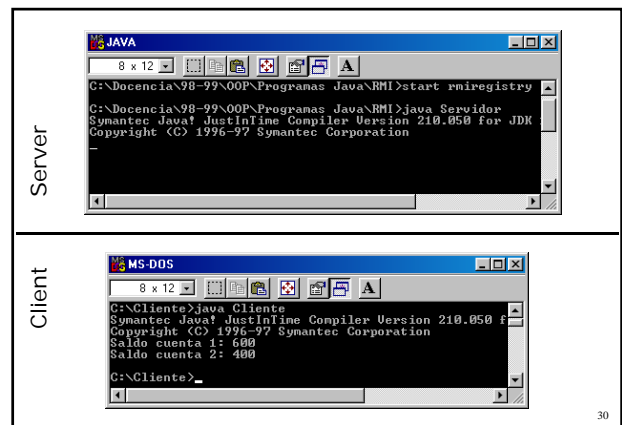
### Execution

- Start the name server (rmiregistry)
- Start the server
- Start the client(s)
- At runtime, the client must have in the local machine:
  - The .class of each remote interface
  - The .class of each corresponding stub class

28



29



30

## Method Invocation (I)

- Stubs are handled as if they were the remote objects themselves
- Using the stubs the programmer handles the remote objects the same as local ones, with the same syntax
- The client calls methods on the stub, and the stub transfers them (over the net, where appropriate) to the real object in the process where it resides
- The method is executed on the server
- The client receives the return value through the inverse path
- RMI takes care of all the the underlying network communication details

31

## Method Invocation (II)

- The client calls a method un a stub
- The stub packs (serializes) the arguments, sends the call to its local JVM, and remains waiting for the response
- The local JVM initiates a connection with the remote JVM that contains the object, and transmits the call and arguments
- When the call response arrives, the stub unpacks (deserializes) the returned value, if any
- The stub returns the value to the program (this may include exceptions thrown by the method at the server)
- The server may or may not execute each method on an independent thread  
In any case, the server should be thread-safe (e.g. using `synchronized` methods, e.g. in bank account operations)

32

## Stubs

- Once the remote classes are compiled, the stub classes are automatically generated by: `rmic xxx.class → xxx_stub.class`
- The implementation of the methods in the stub class is different from the implementation of the same methods in the remote class
  - Method in the stub: client-server mediation, packs the arguments and transmits the call to the server
  - Method in the server: holds the real semantic of the method
- The stub contains information about the object it represents
  - Object location: host and port
  - Object class
  - An ID that identifies each remote object on the server
- The stub hides
  - Serialization
  - Details of the communication with the server

33

## Internal Management of the Communication between Processes

RMI takes care of:

- Opening, controlling and closing connections between client and server
- Packing and unpacking data
- Prepare the server to remain waiting for method calls from clients using sockets
- Take care of the calls dispatching them to the corresponding objects

34

## Remote Classes `UnicastRemoteObject`

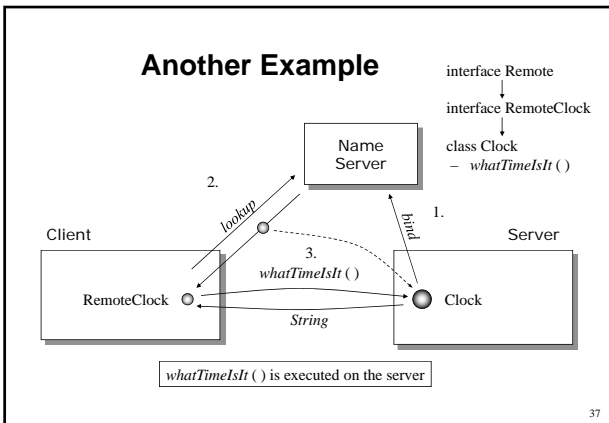
- `java.rmi.server.UnicastRemoteObject`
  - Calls `UnicastRemoteObject.exportObject(this)` in its constructor
  - Redefines some methods of `Object`: `equals`, `hashCode`, `toString`
- It is not mandatory to extend `UnicastRemoteObject`
  - It is enough to call `exportObject(obj[, puerto])` on the instances (e.g. in the constructor)
- `exportObject(Remote)`
  - Registers the existence of the object in the RMI runtime system
  - The object remains waiting for calls from clients using an anonymous port chosen by RMI or the operating system
- `getClientHost()` returns the client IP address

35

## Server Life Cycle

- Main creates objects and registers them in the name server
- Main terminates but the server remains active: the JVM keeps running
- The RMI system keeps a server active as long as some reference remains to some of the objects created in the server
- This includes the references (stubs) to the objects from the name server
- When an object has no longer references to it from any client or within the server itself, it becomes available for GC

36



### RemoteClock.java

```

import java.rmi.*;

public interface RemoteClock extends Remote {
    public String whatTimeIsIt () throws RemoteException;
}
    
```

38

### Server.java

```

import java.rmi.*;
import java.rmi.server.*;
import java.util.*;
import java.text.*;

class Clock extends UnicastRemoteObject
    implements RemoteClock {
    private String zone;
    private DateFormat timeFormat;
    Clock (String zone) throws RemoteException {
        this.zona = zone;
        timeFormat = DateFormat.getTimeInstance ();
        timeFormat.setTimeZone (TimeZone.getTimeZone (zone));
    }
    public String whatTimeIsIt () throws RemoteException {
        String time = timeFormat.format (new Date ());
        System.out.println ("Client asks for the time in "
            + zone + ": " + time);
        return time;
    }
}
    
```

39

```

public class Server {
    public static void main (String args[]) throws Exception {
        Clock peninsula = new Clock ("Europe/Madrid");
        Naming.rebind ("Peninsular time", peninsula);
        Clock canarias = new Clock ("Europe/Lisbon");
        Naming.rebind ("Canarias time", canarias);
    }
}
    
```

40

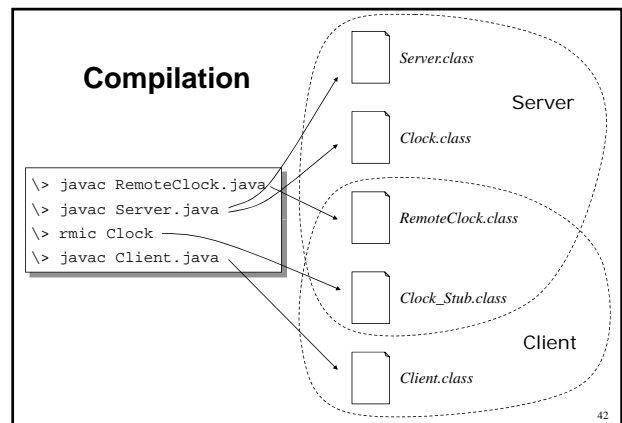
### Client.java

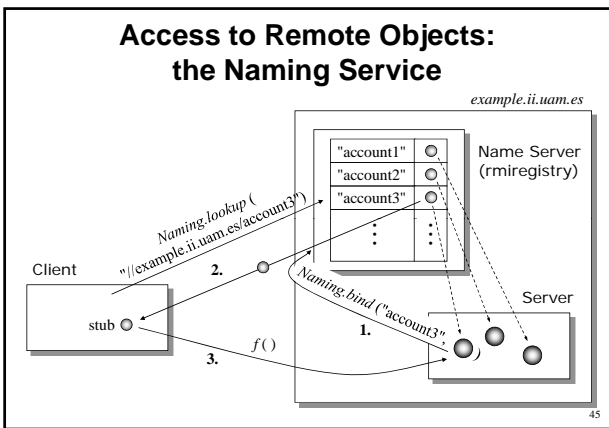
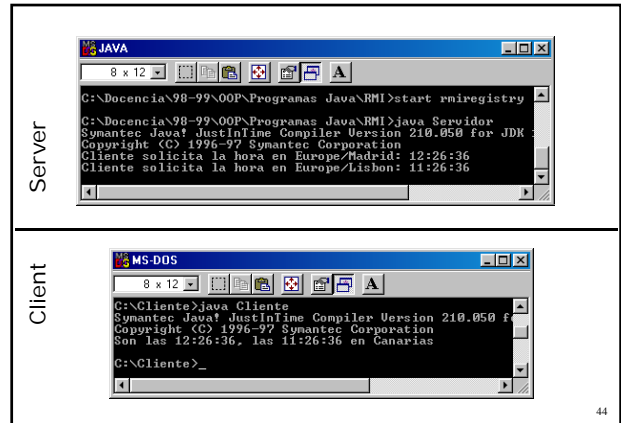
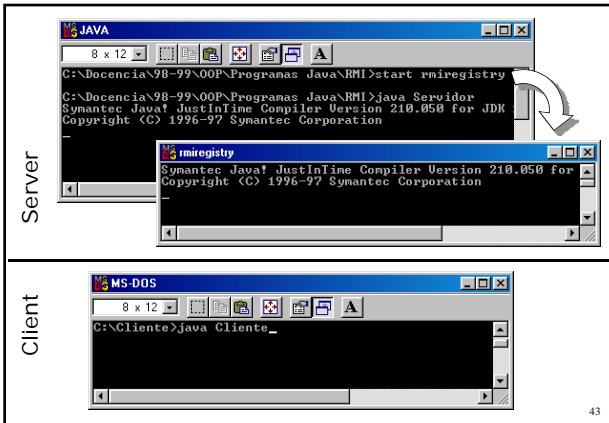
```

import java.rmi.*;

public class Client {
    public static void main (String args[]) throws Exception {
        RemoteClock clock1 = (RemoteClock) Naming.lookup (
            "//example.ii.uam.es/Peninsular time");
        RemoteClock clock2 = (RemoteClock) Naming.lookup (
            "//example.ii.uam.es/Canarias time");
        System.out.println ("It is "
            + clock1.whatTimeIsIt ()
            + ", "
            + clock2.whatTimeIsIt ()
            + " in Canarias");
    }
}
    
```

41



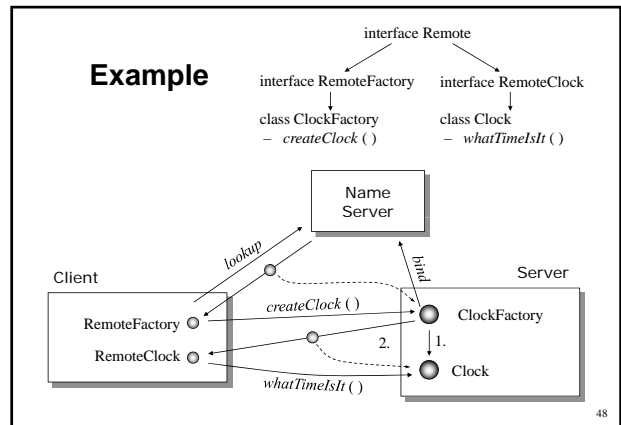


### The Name Server Program: rmiregistry

- By default, rmiregistry uses port 1099
- Several rmiregistries can be started using different ports: `rmiregistry port`
- To rebind and lookup on a rmiregistry with a port different from 1099, use `"/host:port/objectName"` as argument
- A remote object may only register its name on a local rmiregistry, in order to prevent any malicious client from manipulating the name registry

### A Different Way to Get Stubs: Factory Objects

- A common way to get stubs in clients
- Getting stubs may be linked to object creation on request of the client
- Factories: objects with methods to create objects of the different remote object classes on a server
- The server creates a single object, a factory, and assigns it a name
- The client accesses the factory object by name
- The client requests the factory object to create other objects
- The factory creates the objects and returns stubs to the client



### RemoteFactory.java

```
import java.rmi.*;

public interface RemoteFactory extends Remote {
    public RemoteClock createClock (String zone)
        throws RemoteException;
}
```

49

### Server.java

```
class Clock implements RemoteClock {
    ...
}
// rmic Clock ClockFactory

class ClockFactory implements RemoteFactory {
    ClockFactory () throws RemoteException {
        UnicastRemoteObject.exportObject (this);
    }
    public RemoteClock createClock (String zone)
        throws RemoteException {
        return new Clock (zone);
    }
}

public class Server {
    public static void main (String args[] throws Exception {
        ClockFactory factory = new ClockFactory ();
        Naming.rebind ("Clock Factory", factory);
    }
}
```

50

### Client.java

```
import java.rmi.Naming;

public class Client {
    public static void main (String args[]) throws Exception {
        RemoteFactory factory = (RemoteFactory) Naming.lookup (
            "//example.ii.uam.es/Clock Factory");
        RemoteClock clock1 = factory.createClock ("Europe/Madrid");
        RemoteClock clock2 = factory.createClock ("Europe/Lisbon");
        System.out.println ("It is "
            + clock1.whatTimeIsIt ()
            + ", "
            + clock2.whatTimeIsIt ()
            + " in Canarias");
    }
}
```

51

### Object Pass by Value

- Local objects may be passed as arguments to remote methods, or returned by remote methods as return value
- A copy of the object is sent
  - The object is serialized in the program of origin as a stream of bytes
  - The program that receives the object restores the object from the byte stream
  - RMI takes care of the serialization in a transparent way for the programmer
- Remote objects are passed differently: by reference instead of by value
  - Remote objects are converted into stubs when passed between programs
- ¿Can any object be passed by value?
  - Only objects of classes that implement `java.io.Serializable`

52

### Object Serialization

- Object packing in form of a stream of bytes
- `java.io.Serializable` does not declare `any` method
  - It is just about marking classes as serializable
  - No new functionality needs to be added
- The serialization functionality is generic and is contained in the `Object{Input|Output}Stream` classes and the corresponding default `{Read|Write}Object` methods
- Default serialization stores:
  - Information about the object class and the host where it is located
  - Variable values that are neither `static` nor `transient` (recursively serialized if they are objects themselves)
- To change default serialization: `readObject(ObjectInputStream)`, `writeObject(ObjectOutputStream)`
- ¿Java standard library classes are serializable?
  - Most are, some are not (e.g. `java.io.FileDescriptor`)

53

### Manipulation of Objects Passed by Value at the Receiving Program

- For object pass by value to make sense, the receiving program must be able to manipulate the objects, i.e. able to invoke their methods
- This means the receiving program includes (in advance) some code where the methods are called on objects passed by value
- The receiver may have the method implementation code, or it may load the class dynamically (it gets the class from the host of origin)
- If the receiver does not have the classes, it is necessary at least to define an interface for these objects, shared by both sender and receiving programs
- This way a client may transfer to the server the execution of arbitrary programs defined at the client, as long as they comply with an interface

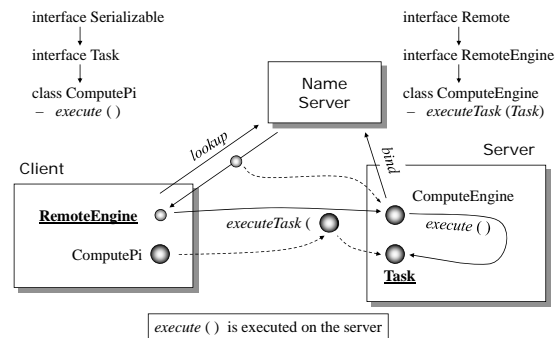
54

### Remote Access to Classes

- A program may get automatically through the web the code of the classes of serializable objects received from a remote program
- For this purpose, the host of origin must have an HTTP server running
- It is also necessary to start the program of origin with the parameter `-Djava.rmi.server.codebase=file: /<directory path>` Indicating the directory where the .class files can be found
- The receiving program loads the classes automatically when the objects are received
- It is also possible to use this mechanism for a client to get remote object stub classes dynamically at runtime

55

### Serialized Objects: Example



56

### RemoteEngine.java

```
import java.rmi.*;

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

### Task.java

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

57

### ComputeEngine.java

```
import java.rmi.*;
import java.rmi.server.*;

public class ComputeEngine extends UnicastRemoteObject
    implements RemoteEngine {
    public ComputeEngine () throws RemoteException {}
    public Object executeTask (Task t)
        throws RemoteException {
        return t.execute ();
    }
}
```

58

### Server.java

```
import java.rmi.*;

public class Server {
    public static void main(String[] args) {
        try {
            ComputeEngine engine = new ComputeEngine ();
            Naming.rebind ("Compute engine", engine);
            System.out.println ("Engine ready");
        } catch (Exception e) { e.printStackTrace(); }
    }
}
```

59

### ComputePi.java

```
public class ComputePi implements Task {
    private int decimals;
    public ComputePi (int n) { decimals = n; }
    public Object execute () {
        double pi = 0, i = 0, term;
        do {
            term = 4 * Math.pow(-1,i) * 1/(2*i+1);
            pi += term;
            i++;
        } while (Math.abs (term) >
            Math.pow (10, -decimals-1));
        return new Double (pi);
    }
}
```

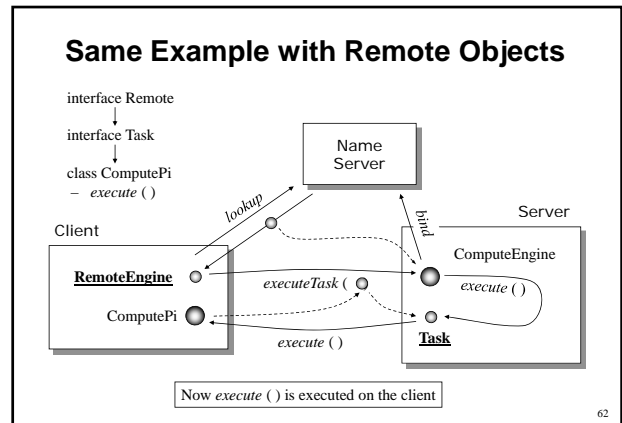
60

### Client.java

```
import java.rmi.*;

public class Client {
    public static void main (String args[]) {
        try {
            RemoteEngine engine = (RemoteEngine) Naming.lookup (
                "//example.ii.uam.es/Compute engine");
            ComputePi task = new ComputePi (4);
            Double pi = (Double) (engine.executeTask (task));
            System.out.println (pi);
        } catch (Exception e) { e.printStackTrace(); }
    }
}
```

61



### Task.java

```
import java.rmi.*;

public interface Task extends Remote {
    Object execute () throws RemoteException;
}
```

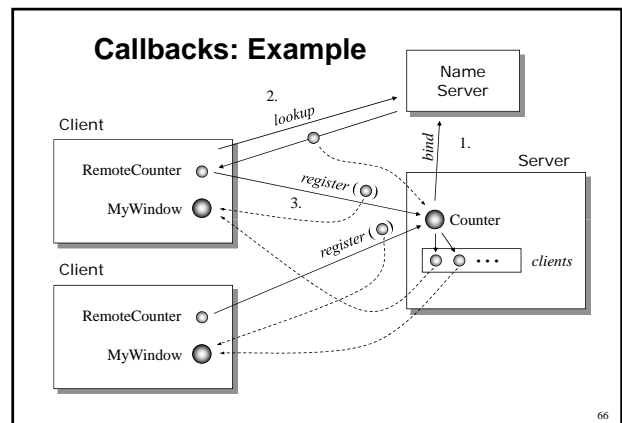
63

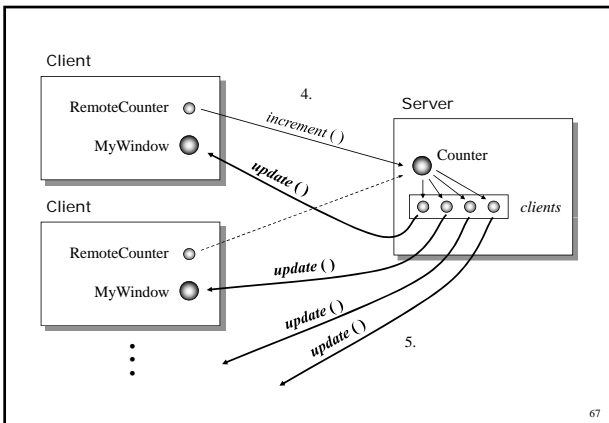
### ComputePi.java

```
public class ComputePi extends UnicastRemoteObject
    implements Task {
    private int decimals;
    public ComputePi (int n) throws RemoteException {
        decimals = n;
    }
    public Object execute () throws RemoteException {
        double pi = 0, i = 0, term;
        do {
            term = 4 * Math.pow(-1,i) * 1/(2*i+1);
            pi += term; i++;
        }
        while (Math.abs (term) > Math.pow (10,-decimals-1));
        return new Double (pi);
    }
}
```

64

- ### Callbacks
- The server acts as a client with respect to certain classes of its clients
  - The server gets a stub to some remote object of the client
  - Usually the client provides this remote object passing it as argument of a method of some remote object of the server
  - The server invokes methods on the remote object of the client
  - The call from the server is nested within the call from the client
  - Usefulness: notification mechanisms from a server to its clients
- 65





```

RemoteCounter.java
import java.rmi.*;

public interface RemoteCounter extends Remote {
    public void increment ()
        throws RemoteException;

    public void registerClient (RemoteWindow v)
        throws RemoteException;

    public void removeClient (RemoteWindow v)
        throws RemoteException;
}

RemoteWindow.java
import java.rmi.*;

public interface RemoteWindow extends Remote {
    public void update (int n)
        throws RemoteException;
}
    
```

```

import java.rmi.*; import java.rmi.server.*;
import java.awt.*; import java.awt.event.*;

public class MyWindow extends Frame MyWindow.java
    implements ActionListener, RemoteWindow {
    private Label label;
    private RemoteCounter counter;
    public MyWindow () throws Exception {
        add (label = new Label ("", Label.CENTER));
        Button b = new Button ("Increment");
        add ("South", b);
        b.addActionListener (this);
        this.addWindowListener (new CloseWindow ());
        UnicastRemoteObject.exportObject (this);
        counter = (RemoteCounter) Naming.lookup (
            "//example.ii.uam.es/Counter Service");
        counter.registerClient (this);
    }
    ...
}
    
```

```

...
public void update (int n) throws RemoteException {
    label.setText ("Value: " + n);
}

public void actionPerformed (ActionEvent e) {
    try { counter.increment (); }
    catch (RemoteException ex) { ex.printStackTrace (); }
}

class CloseWindow extends WindowAdapter {
    public void windowClosing (WindowEvent e) {
        MyWindow v = (MyWindow) e.getSource ();
        try { counter.removeClient (v); System.exit(0); }
        catch (RemoteException ex) { ex.printStackTrace (); }
    }
}
    
```

```

Client.java

import java.rmi.*;
public class Client {
    public static void main (String args[]) {
        try {
            MyWindow v = new MyWindow ();
            v.setVisible (true);
        } catch (Exception e) { e.printStackTrace (); }
    }
}
    
```

```

Counter.java

import java.rmi.*;
import java.rmi.server.*;
import java.util.*;

public class Counter extends UnicastRemoteObject
    implements RemoteCounter {
    private int counter = 0;
    private ArrayList clients = new ArrayList ();
    public Counter () throws RemoteException { }

    public synchronized void increment ()
        throws RemoteException {
        System.out.println ("Current value: " + ++counter);
        Iterator windows = clients.iterator ();
        while (windows.hasNext ())
            ((RemoteWindow) windows.next ())
                .update (counter);
    }
    ...
}
    
```

```

...
public void registerClient (RemoteWindow window)
    throws RemoteException {
    clients.add (window);
    window.update (counter);
}

public void removeClient (RemoteWindow window)
    throws RemoteException {
    clients.remove (window);
}

```

73

### Server.java

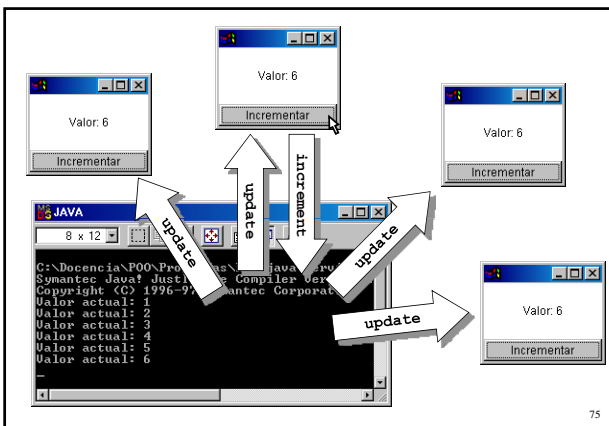
```

import java.rmi.*;

public class Server {
    public static void main (String[] args) {
        try {
            Counter c = new Counter ();
            Naming.rebind ("Counter Service", c);
        } catch (Exception e) { e.printStackTrace (); }
    }
}

```

74



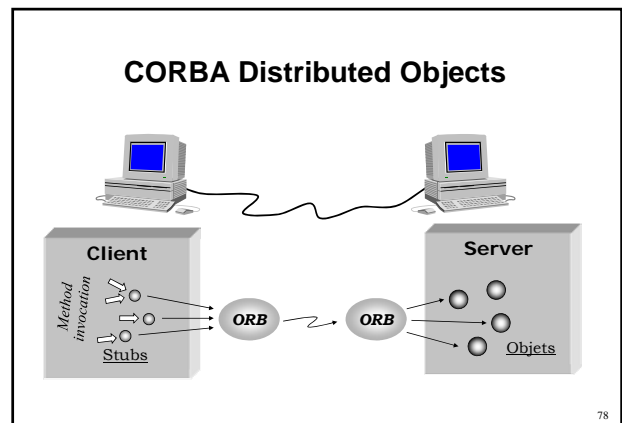
75

## Distributed Objects

### Common Object Request Broker Architecture (CORBA)

- ### Common Object Request Broker Architecture
- Similar to RMI, standard for distributed objects
  - Allows the integration of programs written in different languages
    - Java, C++, C, Smalltalk, Ada, COBOL, etc.
  - Provides additional integrated services
    - Naming, persistence, events, etc.
  - Created by the OMG (Object Management Group)
    - OMG = over 800 empresas: Sun, IBM, Apple, Digital, Netscape, Oracle, Borland, NCR, Siemens, ILOG, Telefónica I+D...
    - OMG founded in 1988
    - CORBA 1.0: 1992, CORBA 2.0: 1997, CORBA 2.6: dic 2001
  - JDK 1.2 / 1.4 incorporates CORBA 2.0 / 2.3 under the name of Java IDL
  - Other available standards: CORBA / RMI / DCOM competition

77

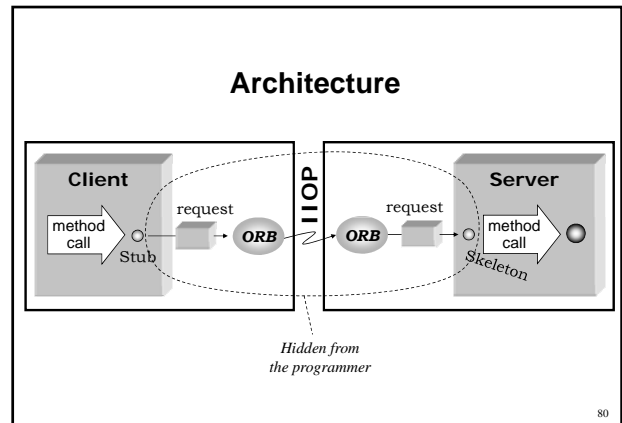


78

### CORBA Architecture

- ORB: mediator between clients and servers
  - Transmits client-server requests and server-client responses
  - An ORB is needed on each machine (in Java IDL, one in each process)
- Stub: mediator between clients and ORB
  - Gets method calls from the client and transfers them to the ORB
  - A stub class for each remote class
- Skeleton: mediator between ORB and server objects
  - Gets method calls from the ORB and executes the methods on the corresponding object in the server

79



### Interface Definition: the IDL Language

- Interface Definition Language
- Descripción de objetos' interface: operations and argument types
- Equivalent to the definition of Remote interfaces in RMI
- Compatible with different programming languages
- Concepts and syntax similar to C++ / Java
- IDL allows inheritance between interfaces
- IDL does not support overriding methods or inherited attributes
- IDL does not support object pass by value (only structs)

81

### Example

```

interface Customer {
    attribute string name;
    attribute long ssn;
};

interface Account {
    attribute long balance;
    attribute long number;
    attribute Customer holder;
    void deposit (in long amount);
    void withdrawal (in long amount);
};

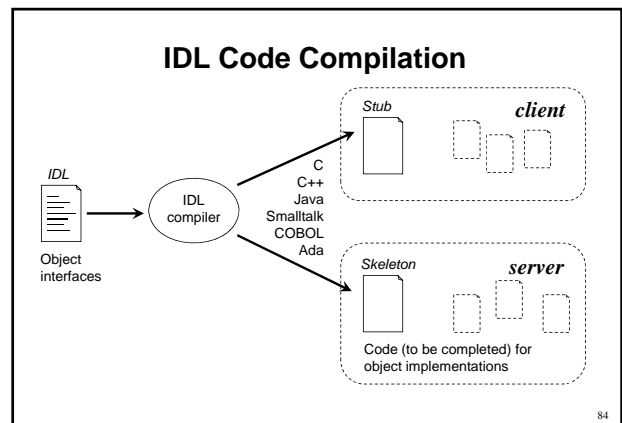
interface CheckingAccount : Account {
    ...
};
    
```

82

### IDL Types

- Basic types: long, short, float, char, boolean, enum, string, any
- Composite types: struct, union
- Derived types: sequence <type>
- Object types: interface

83



IDL Construct	Java Construct
module	package
interface	interface, helper and holder class
constant	public static final
boolean	boolean
char, wchar	char
octet	byte
string, wstring	java.lang.String
short, unsigned short	short
long, unsigned long	int
long long	long
unsigned long long	long
float	float
double	double
enum, struct, union	final class
sequence, array	array
exception	final class
readonly attribute	method for accessing attribute
readwrite attribute	methods for accessing and setting attribute
operation	method

Mapping IDL → Java

## CORBA Implementations

A CORBA implementation includes:

- Components
  - An IDL compiler to different programming languages
  - Libraries for the generation and transmission of messages between processes when clients invoke an object's methods
  - An ORB: may be a program, a DLL, or a class
- At design level
  - Details not defined by the standard
  - Version-specific extensions

## CORBA Products

- Commercial versions:
  - Iona: Orbix
  - Visigenic: Visibroker
  - Digital: ObjectBroker
  - HP: ORB Plus
  - Chorus Systems: CHORUS/COOL ORB
  - IBM: ComponentBroker
- Free versions:
  - Object Oriented Concepts: ORBacus
  - Olivetti Research Lab: OmniORB
  - Sun: Java IDL
- See also:
  - <http://www.cs.wustl.edu/~schmidt/corba-products.html>
  - <http://adams.patriot.net/~tvalesky/freecorba.html>

## Compatibilities and Incompatibilities

1. Different ORB versions can communicate between each other (IIOP)
2. The code written for an ORB version cannot directly communicate with other ORB versions
3. Applications developed for a CORBA version in different programming languages, OS and hardware, can be integrated through a shared ORB

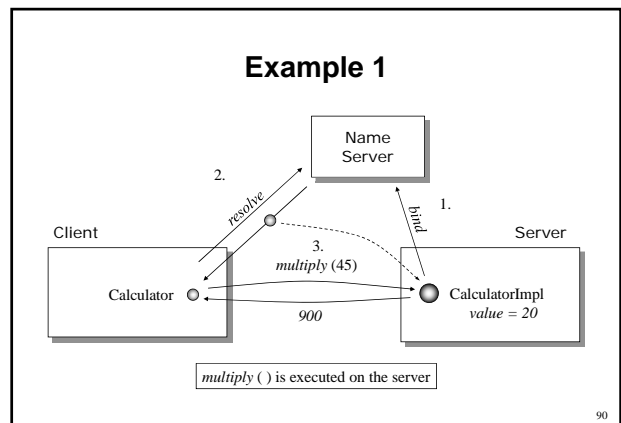
## CORBA vs. RMI

CORBA

- Integration of legacy applications
- Heterogeneity may be desirable: different languages for different needs
- Integrated services

RMI

- Ease of programming, portability of implementations
- Object pass by value
- Java includes in the language specification:
  - Web programming
  - User interface programming
  - And more...
- RMI + JNI permite integrar distintos lenguajes, sin embargo...
  - La integración es a nivel de llamadas a funciones, no de objetos
  - Programación mucho más tediosa que CORBA
- JDK 1.3 adopta IIOP

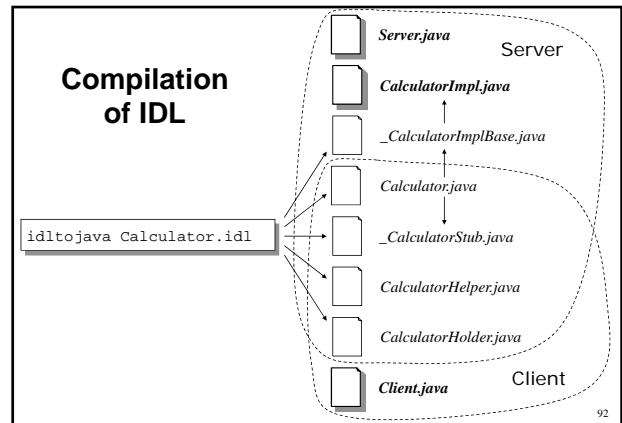


### Calculator.idl

```

interface Calculator {
    attribute double value;
    double add (in double x);
    double multiply (in double x);
    double sqroot ();
    void reset ();
};
    
```

91



### Calculator.java

```

//-----
// Code generated by idltojava
//-----

public interface Calculator
    extends org.omg.CORBA.Object,
           org.omg.CORBA.portable.IDLEntity {

    double value ();
    void value (double arg);
    double add (double n);
    double multiply (double n);
    double sqroot ();
    void reset ();
}
    
```

93

### \_CalculatorImplBase.java

```

//-----
// Code generated by idltojava
//-----

public abstract class _CalculatorImplBase
    extends org.omg.CORBA.DynamicImplementation
    implements Calculator {

    ...
}
    
```

94

### CalculatorImpl.java

```

import org.omg.CORBA.*;

class CalculatorImpl extends _CalculatorImplBase {
    private double value;
    public double value () { return value; }
    public void value (double x) { value = x; }
    ...
}
    
```

95

```

...
public double add (double x) {
    value += x;
    System.out.println ("+ " + x + " = " + value);
    return value;
}
public double multiply (double x) {
    value *= x;
    System.out.println ("* " + x + " = " + value);
    return value;
}
public double sqroot () {
    value = Math.sqrt (value);
    System.out.println ("sqroot = " + value);
    return value;
}
public void reset () { value = 0; }
}
    
```

96

```

import org.omg.CORBA.*;
import org.omg.CosNaming.*;

public class Server {
    public static void main (String args[] ) {
        try {
            ORB orb = ORB.init (args, null);
            CalculatorImpl calc = new CalculatorImpl ();
            orb.connect (calc);
            System.out.println ("Calculator ready\nValue: " +
                calc.value ());

            NamingContext naming = NamingContextHelper.narrow (
                orb.resolve_initial_references ("NameService"));
            NameComponent name[] =
                { new NameComponent ("My calculator", "") };
            naming.rebind (name, calc);

            java.lang.Object sync = new java.lang.Object ();
            synchronized (sync) { sync.wait(); }
        } catch (Exception e) { e.printStackTrace (); }
    }
}

```

97

```

import org.omg.CORBA.*;
import org.omg.CosNaming.*;

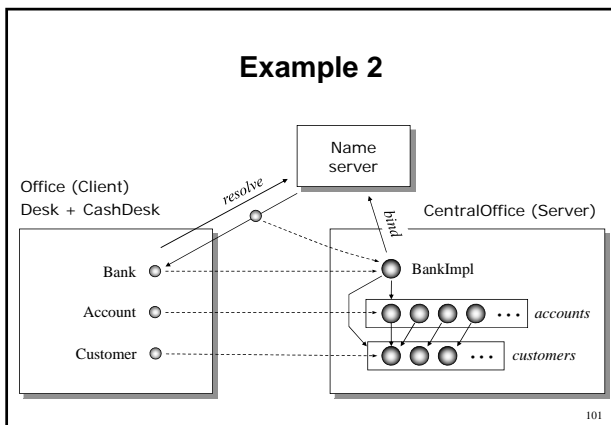
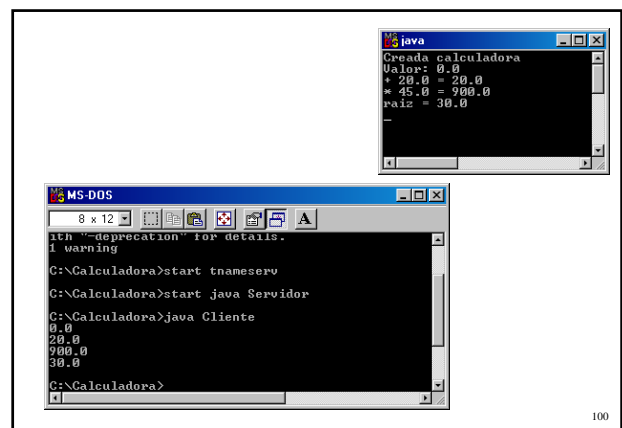
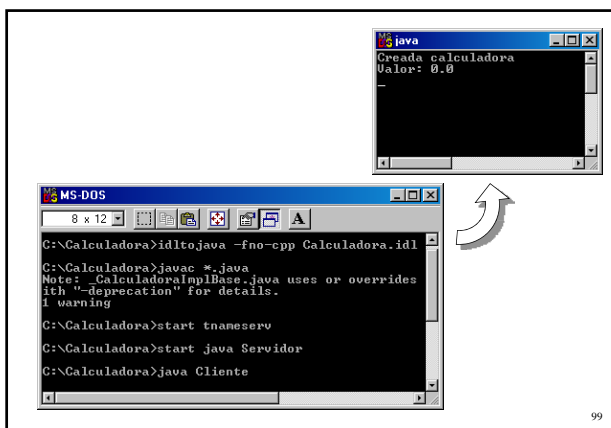
public class Client {
    public static void main (String args [] ) {
        try {
            ORB orb = ORB.init (args, null);

            NamingContext naming = NamingContextHelper.narrow (
                orb.resolve_initial_references ("NameService"));
            NameComponent name[] =
                { new NameComponent ("My calculator", "") };
            Calculator calc = CalculatorHelper.narrow (
                naming.resolve (name));

            System.out.println (calc.value ());
            calc.add (20); System.out.println (calc.value ());
            calc.multiply (45); System.out.println (calc.value ());
            calc.root (); System.out.println (calc.value ());
        } catch (Exception e) { e.printStackTrace (); }
    }
}

```

98



```

interface Customer {
    attribute string name;
    attribute long ssn;
};

interface Account {
    exception InsufficientBalance { long balance; };
    attribute long balance;
    attribute long number;
    attribute Customer holder;
    void deposit (in long amount);
    void withdrawal (in long amount)
        raises (InsufficientBalance);
};

interface Bank {
    attribute string name;
    Account openAccount (in Customer holder);
    Account getAccount (in long number);
    void cancelAccount (in long number);
    Customer newCustomer (in string name, in long ssn);
    Customer getCustomer (in long ssn);
    void dropCustomer (in long ssn);
};

```

102

### InsufficientBalance.java

```
//-----  
// Code generated by idltojava  
//-----  
  
package AccountPackage;  
  
public final class InsufficientBalance  
    extends org.omg.CORBA.UserException  
    implements org.omg.CORBA.portable.IDLEntity {  
    public int balance;  
    public InsufficientBalance () { super (); }  
    public InsufficientBalance (int __balance) {  
        super ();  
        balance = __balance;  
    }  
}
```

103

### BankImpl.java

```
import org.omg.CORBA.*;  
import java.util.*;  
  
public class BankImpl extends _BankImplBase {  
    ORB orb;  
    private String name;  
    private ArrayList accounts = new ArrayList ();  
    private ArrayList customers = new ArrayList ();  
    BankImpl (String str, ORB orb) {  
        name = str;  
        this.ORB = orb;  
        orb.connect (this);  
    }  
  
    public String name () { return name; }  
    public void name (String str) { name = str; }  
    ...  
}
```

104

```
...  
public Account openAccount (Customer holder) {  
    AccountImpl account = new AccountImpl (holder, orb);  
    accounts.add (account);  
    return account;  
}  
  
public Account getAccount (int number) {  
    Iterator iter = accounts.iterator ();  
    while (iter.hasNext ()) {  
        Account account = (Account) iter.next ();  
        if (account.number () == number) return account;  
    }  
    return null; // Exception  
}  
  
public void cancelAccount (int number) {  
    accounts.remove (getAccount (number));  
    ...  
}
```

105

```
...  
public Customer newCustomer (String name, int ssn) {  
    CustomerImpl customer =  
        new CustomerImpl (name, ssn, orb);  
    customers.add (customer);  
    return customer;  
}  
  
public Customer getCustomer (int ssn) {  
    Iterator iter = customers.iterator ();  
    while (iter.hasNext ()) {  
        Customer customer = (Customer) iter.next ();  
        if (customer.ssn () == ssn) return customer;  
    }  
    return null; // Exception  
}  
  
public void dropCustomer (int ssn) {  
    customers.remove (getCustomer (ssn));  
};
```

106

### AccountImpl.java

```
import org.omg.CORBA.*;  
  
public class AccountImpl  
    extends _AccountImplBase {  
    private static int naccounts = 0;  
    private int number, balance;  
    private Customer holder;  
    AccountImpl (Customer clt, ORB orb) {  
        number = ++naccounts;  
        balance = 0; holder = clt;  
        orb.connect (this);  
    }  
  
    public int balance () { return balance; }  
    public void balance (int n) { balance = n; }  
    public int number () { return number; }  
    public void number (int n) { number = n; }  
    public Customer holder () { return holder; }  
    public void holder (Customer clt) { holder = clt; }  
    ...  
}
```

107

```
...  
public synchronized void deposit (int amount) {  
    balance += amount;  
}  
  
public synchronized void withdrawal (int amount)  
    throws AccountPackage.InsufficientBalance {  
    if (balance < amount)  
        throw  
            new AccountPackage.InsufficientBalance (balance);  
    balance -= amount; }  
}
```

108

```

CustomerImpl.java

import org.omg.CORBA.*;

public class CustomerImpl extends _CustomerImplBase {
    private String name;
    private int ssn;
    CustomerImpl (String str, int n, ORB orb) {
        name = str;
        ssn = n;
        orb.connect (this);
    }

    public String name () { return name; }
    public void name (String str) { name = str; }
    public int ssn () { return ssn; }
    public void ssn (int n) { ssn = n; }
}
    
```

109

```

CentralOffice.java (server)

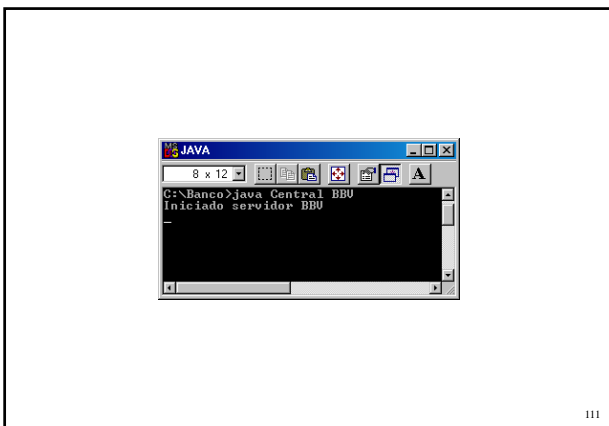
import org.omg.CORBA.*;
import org.omg.CosNaming.*;

public class CentralOffice {
    public static void main (String args[]) {
        try {
            ORB orb = ORB.init (args, null);
            BankImpl bank = new BankImpl (args[0], orb);

            NamingContext naming = NamingContextHelper.narrow (
                orb.resolve_initial_references ("NameService"));
            NameComponent name[] =
                { new NameComponent (args[0], "") };
            naming.rebind (name, bank);

            System.out.println ("Server started " + args[0]);
            java.lang.Object sync = new java.lang.Object ();
            synchronized (sync) { sync.wait(); }
        } catch (Exception e) { e.printStackTrace (); }
    }
}
    
```

110



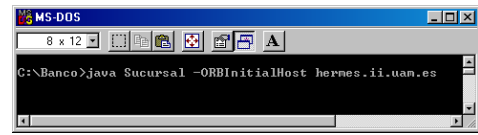
```

Office.java (main client)

import org.omg.CORBA.*;

public class Office {
    public static void main (String args []) {
        try {
            ORB orb = ORB.init (args, null);
            new Desk (orb) .setVisible (true);
            new CashDesk (orb) .setVisible (true);
        } catch (Exception e) { e.printStackTrace (); }
    }
}
    
```

112



```

import org.omg.CORBA.*; import org.omg.CosNaming.*;
import java.awt.*; import java.awt.event.*;

public class Desk extends Frame implements ActionListener {
    TextField bankName, holderName, holderSSN;
    Label accountNumber, accountBalance;
    Bank bank;
    Account account;
    NamingContext naming;
    Desk (ORB orb) throws Exception {
        setTitle ("Desk");
        add (new Label ("Bank name"));
        add (bankName = new TextField ("", 30));
        add (new Label ("Account holder name"));
        add (holderName = new TextField (""));
        add (new Label ("Holder SSN"));
        add (holderSSN = new TextField (""));
        add (new Label ("Account Nr:"));
        add (accountNumber = new Label (""));
        add (new Label ("Balance:"));
        add (accountBalance = new Label (""));
        ...
    }
}
    
```

**Desk.java (client)**  
113

```

...
Panel buttonPanel = new Panel ();
add (buttonPanel);
Button b = new Button ("Create");
buttonPanel.add (b);
b.addActionListener (this);
b = new Button ("Close");
buttonPanel.add (b);
b.addActionListener (this);
... // definir layout

naming = NamingContextHelper.narrow (
    orb.resolve_initial_references ("NameService"));
}

void connectToBank () throws Exception {
    NameComponent name[] =
        { new NameComponent (bankName.getText (), "") };
    bank = BankHelper.narrow (naming.resolve (name));
}
...
    
```

114

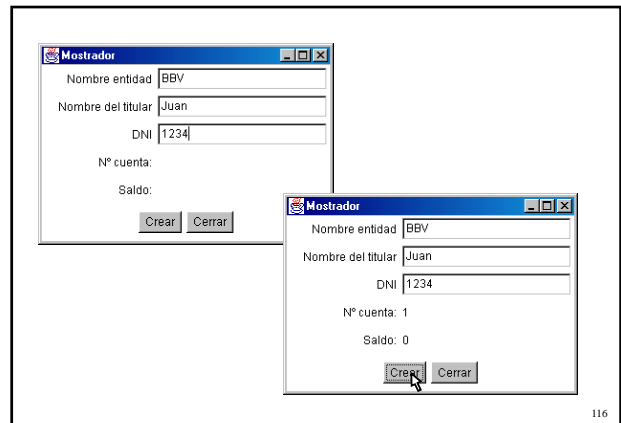
```

...
public void actionPerformed (ActionEvent e) {
    if (e.getActionCommand () == "Close") dispose ();
    else try {
        connectToBank ();

        String name = holderName.getText ();
        int ssn = Integer.valueOf (holderSSN.getText ())
            .intValue ();
        Customer holder = bank.getCustomer (ssn);
        if (holder == null)
            holder = bank.newCustomer (name, ssn);

        account = bank.openAccount (holder);
        holderName.setText (account.holder ().name ());
        accountNumber.setText (
            String.valueOf (account.number ());
        accountBalance.setText (
            String.valueOf (account.balance ());
    } catch (Exception ex) { ex.printStackTrace (); }
}
}

```



### CashDesk.java (client)

```

import org.omg.CORBA.*; import org.omg.CosNaming.*;
import java.awt.*; import java.awt.event.*;

public class CashDesk extends Frame
    implements ActionListener {
    TextField bankName, accountNumber, amount;
    Label holderName, holderSSN, accountBalance;
    Bank bank;
    Account account;
    NamingContext naming;
    ...
}

```

```

...
CashDesk (ORB orb) throws Exception {
    setTitle ("Cash Desk");
    add (new Label ("Bank name"));
    add (bankName = new TextField ("", 30));
    add (new Label ("Account Nr"));
    add (accountNumber = new TextField (""));
    add (new Label ("Amount"));
    add (amount = new TextField ("0"));
    add (new Label ("Account holder name:"));
    add (holderName = new Label (""));
    add (new Label ("Holder SSN:"));
    add (holderSSN = new Label (""));
    add (new Label ("Balance:"));
    add (accountBalance = new Label (""));
    ...
}

```

```

...
Panel buttonPanel = new Panel ();
addPart (buttonPanel, layout, c);
Button b = new Button ("Balance");
buttonPanel.add (b);
b.addActionListener (this);
b = new Button ("Deposit");
buttonPanel.add (b);
b.addActionListener (this);
b = new Button ("Withdrawal");
buttonPanel.add (b);
b.addActionListener (this);
b = new Button ("Close");
buttonPanel.add (b);
b.addActionListener (this);
... // Define layout

naming = NamingContextHelper.narrow (
    orb.resolve_initial_references ("NameService"));
} // End constructor
...

```

```

...
void connectToBank () throws Exception {
    NameComponent name[] =
        { new NameComponent (bankName.getText (), "") };
    bank = BankHelper.narrow (naming.resolve (name));
}

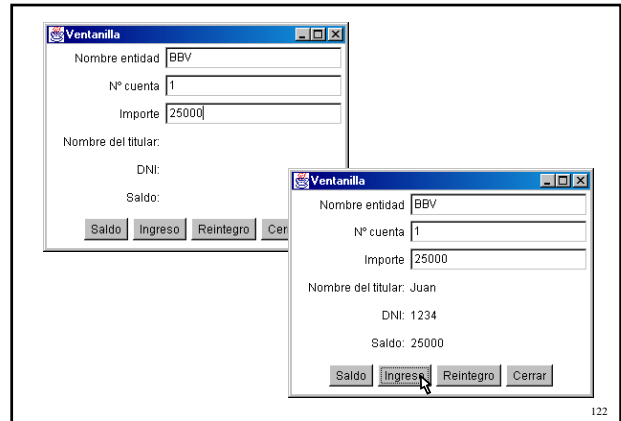
public void actionPerformed (ActionEvent e) {
    String command = e.getActionCommand ();
    if (command == "Close") dispose ();
    else try {
        connectToBank ();
        int number = Integer.valueOf (accountNumber.getText ())
            .intValue ();
        account = bank.getAccount (number);
        holderName.setText (account.holder ().name ());
        holderSSN.setText (String.valueOf (
            account.holder ().ssn ());
    }
    ...
}

```

```

...
int amount =
    Integer.valueOf (amount.getText ())
        .intValue ();
if (command == "Deposit")
    account.deposit (amount);
else if (command == "Withdrawal")
    account.withdrawal (amount);
accountBalance.setText (
    String.valueOf (account.balance ()));
} catch (Exception ex) { ex.printStackTrace (); }
}
}
    
```

121



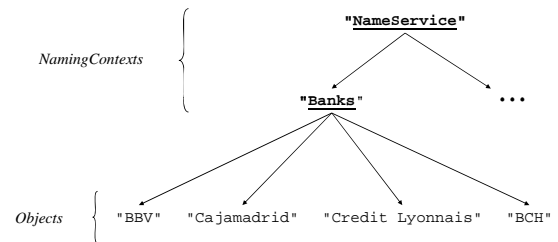
122

### Naming Service

- More sophisticated than that of RMI
- Name = array of names = path in directory-tree-like structure
  - Intermediate array elements are the analogous of directory names: NamingContext's
  - The last array element is the name for an object
- It is necessary to create NamingContexts previously
- Browsing names in the name server
  - Names are often not known in advance
  - It is possible to get a list of all the existing names under a certain NamingContext (like doing a 'dir' of a directory)

123

### Example



124

```

public class CentralOffice {
    public static void main (String args[]) {
        try {
            ORB orb = ORB.init (args, null);
            BankImpl bank = new BankImpl (args[0], orb);

            NamingContext naming = NamingContextHelper.narrow (
                orb.resolve_initial_references ("NameService"));
            NameComponent name[] = {
                new NameComponent ("Banks", "") };
            try { naming.resolve (name); }
            catch (NotFound ex) { naming.bind_new_context (name); }
            NameComponent bankName[] = {
                new NameComponent ("Banks", ""),
                new NameComponent (args[0], "") };
            naming.rebind (bankName, bank);

            System.out.println ("Server started " + args[0]);
            java.lang.Object sync = new java.lang.Object ();
            synchronized (sync) { sync.wait(); }
        } catch (Exception e) { e.printStackTrace (); }
    }
}
    
```

125

### Desk, CashDesk (I)

```

void connectToBank () throws Exception {
    NameComponent bankName[] = {
        new NameComponent ("Banks", ""),
        new NameComponent (bankName.getText (), "") };
    bank = BankHelper.narrow (naming.resolve (bankName));
}
    
```

126

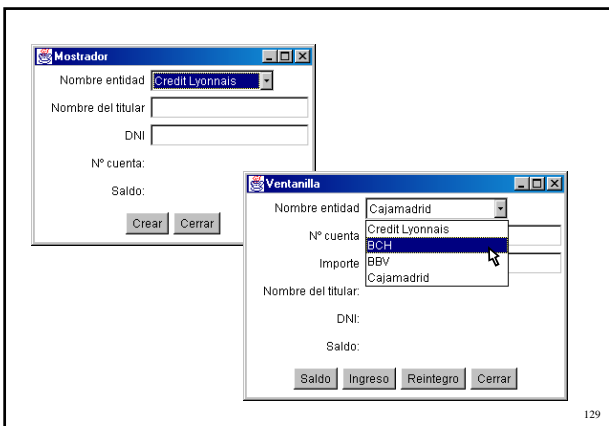
```
public class { Desk | CashDesk }
    extends Frame implements ActionListener {
    Choice bankName;
    ...
    CashDesk (ORB orb) throws Exception {
        ...
        add (bankName = new Choice ());
        ...
        naming = NamingContextHelper.narrow (
            orb.resolve_initial_references ("NameService"));
        NameComponent bankNames[] = {
            new NameComponent ("Banks", "");
        };
        NamingContext banks = NamingContextHelper.narrow (
            naming.resolve (bankNames));
        BindingIteratorHolder iter = new BindingIteratorHolder ();
        BindingListHolder names = new BindingListHolder ();
        banks.list (1000, names, iter);
        for (int i = 0; i < names.value.length; i++) {
            String name = names.value[i].binding_name[0].id;
            bankName.addItem (name);
        }
    }
    ...
}
```

**Desk,  
CashDesk (II)**

127

```
...
void connectToBank () throws Exception {
    NameComponent bankName[] = {
        new NameComponent ("Banks", ""),
        new NameComponent (bankName.getSelectedItem (), "")
    };
    bank = BankHelper.narrow (
        naming.resolve (bankName));
}
...
}
```

128



**Other Forms of Distributed  
Programming in Java**

### URL (Uniform Resource Locator)

- Access to web resources through different protocols
  - ☞ - HTTP (HyperText Transfer Protocol)
  - ftp, gopher, news
  - Connection to DB for JDBC
- Main use
  - Direct reading of remote files
  - Execution and communication with cgi/servlets
- Information described by a URL
  - Protocol name
  - Resource address
    - Host name or IP address
    - Port (optional)
    - File pathname
    - Internal reference within the file (optional)

http : //java.sun.com  
 Protocol Identifier      Resource Name

131

### The java.net.URL Class

- Constructors
  - URL ii = new URL ("http://www.ii.uam.es/esp/welcomii.html");
  - URL courses = new URL (ii, "/esp/alumnos/curso3.html");
  - URL staff = new URL ("http", "www.ii.uam.es",  
"/esp/docentes/index.html");
  - URL uam = new URL ("http://www.uam.es:80/index.html");
  - URL uam = new URL ("http", "www.uam.es", 80, "/index.html");
  - URL timetable = new URL (ii, "esp/alumnos/index.html#horarios");
- They all declare a MalformedURLException
- Parsing
  - getProtocol()
  - getHost()
  - getFile()
  - getRef()
  - getPort()
- Conversion to string:
  - toString ()

132

## Reading and Writing to a URL

- Establish URL connection
  - `url.openConnection()` → `URLConnection`
- Get input / output streams
  - `getInputStream()` → `InputStream`
  - `getOutputStream()` → `OutputStream`
- These methods throw `IOException`

133

## Reading from a URL: Example

```
import java.net.*;
import java.io.*;
public class LeerURL {
    public static void main(String[] args) throws Exception {
        URL uam = new URL ("http://www.ii.uam.es/");
        URLConnection conection = uam.openConnection ();
        BufferedReader input = new BufferedReader (
            new InputStreamReader (conection.getInputStream ());
        String line;
        while ((line = input.readLine()) != null)
            System.out.println(line);
        input.close();
    }
}
```

134

```
Build Successful
<HTML>
<HEAD>
  <TITLE>Escuela T&eacute;cnic;a Superior de
  Inform&aacute;tica</TITLE>
</HEAD>
<BODY BACKGROUND="/imagen/fondouam.gif">
...
<P><B><I>Fecha de &uacute;l;tima actualizaci&oacute;n: 5 de
  Mayo de 1999</I></B></P>
</BODY>
</HTML>
Application terminated
```

135

## Writing to a URL: Example

```
public static void main (String[] args) throws Exception {
    URL sun = new URL ("http://java.sun.com/cgi-bin/backwards");
    URLConnection conection = sun.openConnection();
    conection.setDoOutput (true); // Allow writing
    PrintWriter output =
        new PrintWriter (conection.getOutputStream ());
    output.println ("string=" + agrs[0]);
    output.close ();
    BufferedReader input = new BufferedReader (
        new InputStreamReader (conection.getInputStream ());
    String line;
    while ((line = input.readLine ()) != null)
        System.out.println (line);
    input.close();
}
```

136

## Servlets

- Similar to CGI
  - Much simpler syntax
    - e.g. in C: `getenv (QUERY_STRING)` → `"param1=value1&param2=value2&..."`
  - Need a specific (web) server, e.g. Tomcat (see [www.apache.org](http://www.apache.org)), Jetty, BEA WebLogic, IBM WebSphere, etc. (application servers)
- Servlets are executed on the server
- They are accessed (invoked) through a URL
  - The URL syntax depends on the servlet server
- Included in Java J2EE: packages `javax.servlet...` (not included in J2SE)
- Typically used for dynamic web page generation

137

## Creating a Servlet

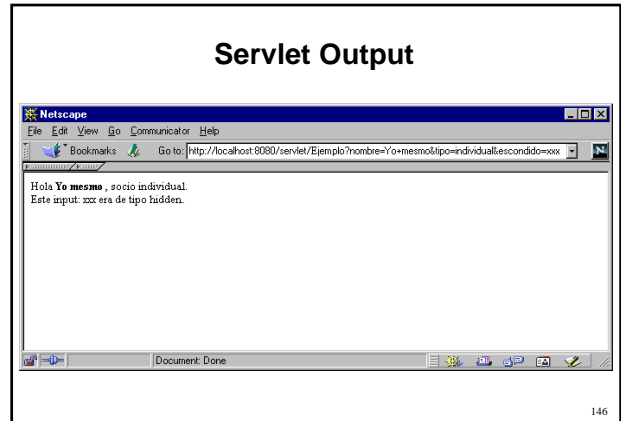
- Define a subclass of `javax.servlet.http.HttpServlet`
- Implement some of the following methods:
  - `doGet (ServletRequest request, ServletResponse response)`
  - `doPost (ServletRequest request, ServletResponse response)`
  - `service (ServletRequest request, ServletResponse response)`
- Register the servlet in the servlet server
  - (Follow instructions in servlets server documentation)

138



```
public class Example extends HttpServlet {
    public void doGet (HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        String name = request.getParameter ("name");
        String type = request.getParameter ("type");
        String hidden = request.getParameter ("hidden");
        PrintWriter out = response.getWriter ();
        out.println (
            "<html><body>"
            + "Hello <b>" + name + "</b>," + type + " member.<br>"
            + "This input: " + hidden + " was of type hidden."
            + "</body></html>"
        );
        out.close ();
    }
}
```

145



- ### JavaServer Pages (JSP)
- Dynamic web page generation
  - Allow combining HTML code and Java code
    - <%= expression %>
    - <% sentence; %>
    - Available implicit variables: request, response, and others
  - Need a web server (application server) that supports JSP, such as Tomcat
  - JSPs are accessed from web browsers just like an HTML page
  - The JSP server compiles the JSP document the first time it is accessed
    - It generates a servlet in a java file (on the server), and compiles it to a .class file
    - The servlet class is loaded and executed (on the server)
  - In fact the output of the JSP does not need to be HTML: it may be XML, etc.
- 147

### Example

example.jsp

```
<html><body>
Hello <b> <%= request.getParameter ("name") %> </b> ,
<%= request.getParameter ("type") %> member. <br>
This input: <%= request.getParameter ("hidden") %>
was of type hidden.
</body></html>
```

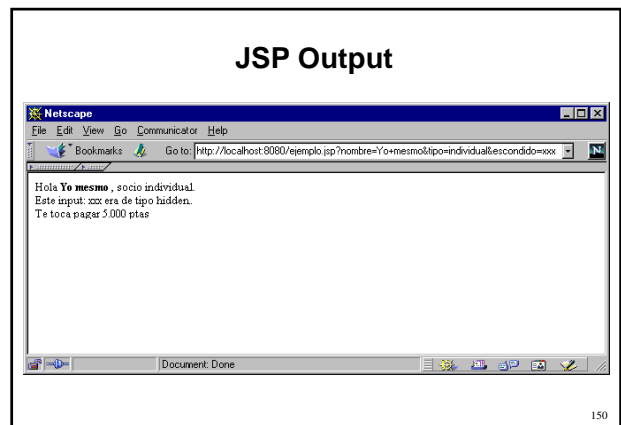
148

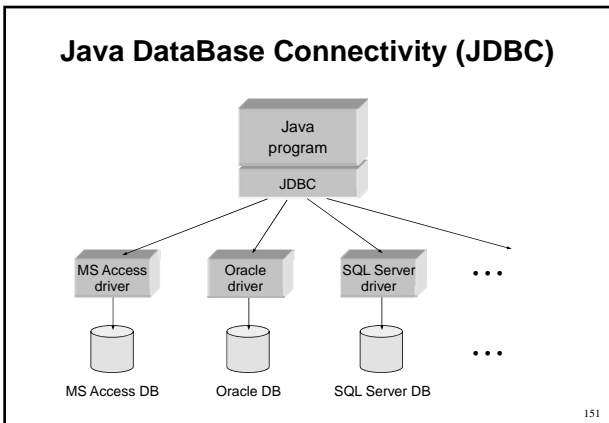
### Control Structures

example.jsp

```
<html><body>
Hello <b> <%= request.getParameter ("name") %> </b> ,
<%= request.getParameter ("type") %> member. <br>
This input: <%= request.getParameter ("hidden") %>
was of type hidden.<br>
<% String type = request.getParameter ("type"); %>
<% if (type.equals ("individual")) { %>
    Your registration fee is 30 euro
<% } else if (type.equals ("student")) { %>
    Your registration fee is 12 euro
<% } else { %>
    Your registration fee is 300 euro
<% } %>
</body></html>
```

149





### JDBC Program Example

```

import java.sql.*;

class Example {
    public static void main (String args[]) throws Exception {
        // Connect to database
        Class.forName ("ids.sql.IDSDriver");
        String url = "jdbc:ids://localhost:12/conn?dsn='example'";
        Connection con = DriverManager.getConnection (url);
        Statement stmt = con.createStatement ();
        ...
    }
}
  
```

152

```

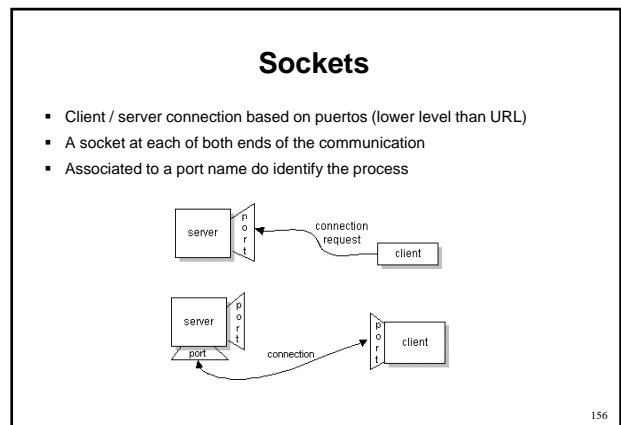
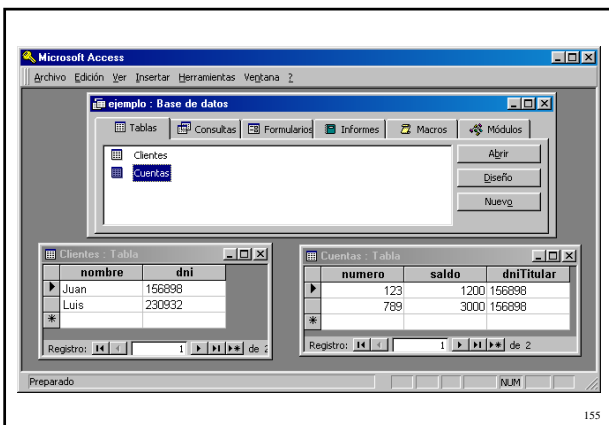
...
stmt.executeUpdate ( // Create Accounts table
    "CREATE TABLE Accounts "
    + "(number INTEGER, balance INTEGER, holderSSN VARCHAR(20))");
stmt.executeUpdate (// Create Customers table
    "CREATE TABLE Customers "
    + "(name VARCHAR(40), ssn VARCHAR(20))");
// Insert a few records into the created tables
stmt.executeUpdate ("INSERT INTO Accounts (number, balance, holderSSN)"
    + " VALUES (123, 1000, '156898')");
stmt.executeUpdate ("INSERT INTO Accounts (number, balance, holderSSN)"
    + " VALUES (456, 500, '230932')");
stmt.executeUpdate ("INSERT INTO Accounts (number, balance, holderSSN)"
    + " VALUES (789, 3000, '156898')");
stmt.executeUpdate ("INSERT INTO Customers (name, ssn)"
    + " VALUES ('John', '156898')");
stmt.executeUpdate ("INSERT INTO Customers (name, ssn)"
    + " VALUES ('Louis', '230932')");
stmt.executeUpdate ("UPDATE Accounts SET balance = balance * 1.2"
    + " WHERE holderSSN = '156898' AND balance < 2000");
stmt.executeUpdate ("DELETE FROM Accounts WHERE balance < 1000");
...
  
```

153

```

...
// Make a query on the database
ResultSet rs = stmt.executeQuery (
    "SELECT Customers.name, Customers.ssn, "
    + "Accounts.number, Accounts.balance "
    + "FROM Accounts, Customers "
    + "WHERE Accounts.holderSSN = Customers.ssn "
    + "AND Accounts.balance > 2000");
while (rs.next ()) {
    String name = rs.getString ("name");
    String ssn = rs.getString ("ssn");
    int number = rs.getInt ("number");
    int balance = rs.getInt ("balance");
    System.out.println (name + " " + ssn + " "
        + number + " " + balance);
} // End main
}
  
```

154



## java.net.Socket java.net.ServerSocket

- Client socket: java.net.Socket
  - Constructor: Socket (String host, int port)  
Generates a local port number, and records the local host IP address
    - getLocalPort(), getLocalAddress()
  - Sending and receiving data (throw IOException)
    - getInputStream()
    - getOutputStream()
- Server socket: java.net.ServerSocket
  - Constructor: ServerSocket (int port)
  - Wait for incoming data: accept() → Socket (throws IOException)
    - The server program stays blocked, waiting for a client to connect
    - When a connection arrives, accept() creates and returns a client socket linked to the incoming client socket
  - Client socket address: getInetAddress(), getPort()

157

## Example: Echo



```
> java EchoClient
Dialogue client/server
hello
(Server) You said: hello
bye
(Server) You said: bye
```

158

## Client

```
import java.io.*; import java.net.*;
public class EchoClient {
    public static void main (String[] args) throws IOException{
        Socket echoSocket = null;
        PrintWriter output = null;
        BufferedReader input = null;
        try {
            echoSocket = new Socket ("example.ii.uam.es", 4444);
            output =
                new PrintWriter (echoSocket.getOutputStream (), true);
            input = new BufferedReader(
                new InputStreamReader (echoSocket.getInputStream ()));
        } catch (UnknownHostException e) {
            System.err.println ("Don't know about host");
        } catch (IOException e) {
            System.err.println ("Couldn't get I/O for connection");
        }
        ...
    }
}
```

159

```
...
BufferedReader stdIn = new BufferedReader(
    new InputStreamReader (System.in));
String userInput;
String fromServer, fromServer;
while ((fromServer = input.readLine ()) != null) {
    System.out.println (fromServer);
    fromUser = stdIn.readLine ();
    output.println (fromUser);
}
// Close streams first, then close sockets
output.close ();
input.close ();
stdIn.close ();
echoSocket.close ();
}
}
```

160

## Server

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

public class EchoServer {
    public static void main (String[] args) throws IOException{
        ServerSocket serverSocket = null;
        try { serverSocket = new ServerSocket (4444); }
        catch (IOException e) {
            System.err.println ("Could not listen on port: 4444.");
        }
        Socket clientSocket = null;
        try { clientSocket = serverSocket.accept (); }
        catch (IOException e) {
            System.err.println ("Accept failed.");
        }
        ...
    }
}
```

161

```
...
BufferedReader input = new BufferedReader (
    new InputStreamReader (clientSocket.getInputStream ()));
PrintWriter output =
    new PrintWriter (clientSocket.getOutputStream (), true);
String line = "Dialogue client/server";
output.println (line);
while ((line = input.readLine ()) != null)
    output.println ("(Server) You said: " + line);
output.close ();
input.close ();
clientSocket.close ();
serverSocket.close ();
}
}
```

162

### Server for Several Simultaneous Sockets

```
...
while (true) {
    Socket clientSocket = null;
    try {
        clientSocket = serverSocket.accept();
        new EchoService (clientSocket) .start ();
    }
    catch (IOException e) {
        System.err.println ("Accept failed.");
    }
}
...
```

163

```
class EchoService extends Thread {
    Socket clientSocket;
    EchoService (Socket s) { clientSocket = s; }
    public void run () {
        try {
            BufferedReader input = new BufferedReader (
                new InputStreamReader (
                    clientSocket.getInputStream ());
            PrintWriter output =
                new PrintWriter (
                    clientSocket.getOutputStream (), true);
            String line = "Dialogue client/server";
            output.println (line);
        }
    }
}
```

164

```
...
while ((line = input.readLine ()) != null)
    output.println ("(Server) You said: " + line);

output.close ();
input.close ();
clientSocket.close ();
} // End try block
catch (IOException e) {
    System.err.println ("Accept failed.");
}
}
}
```

165