

Enterprise Java

“Enterprise” Java

- Group of APIs (class libraries) intended to facilitate large scale business applications, primarily database related applications
 - JDBC Java **D**atabase **C**onnectivity
 - RMI Remote Method Invocation
 - Java IDL Interface Definition Language (CORBA compliance)
 - JNDI Directory and naming services
 - ...

Enterprise Java

- ...
- JNDI (Directory and naming services)
- Servlets & JSPs
- EJBs
- Messaging (JMS)
- WebServices

JDBC



JDBC

- Allow a Java application (or, possibly, Applet) to work with a **relational database**.
- ... *work with a what?*
- Relational databases are covered in later subjects (CSCI235, CSCI315, CSCI317)
 - imagine them as holding many large “tables”
 - columns represent fields
 - rows represent individual records

Customer	Name	Address	Phone #
01278901	Jones, D.	123 Main, ...	01-345-.
01278913	Smith, J.	6 High, ...	01-348-.
01278927	White, P.	42 Market, ..	01-345-.
01278944	Black, T.	3 Olive, ...	02-448-.
01278956	Chung, H.	77 West, ...	01-345-.
01278967	O'Brien, 4	Cliff, ...	01-348-.
01278968	Curry, N.	6/33 Accacia.	-----
01278971	Kovac, J.	7b Chessim, .	02-448-.

Magazine subscription database – table 1, customer names and addresses

Customer Number	Title	Renew date	...
01278901	Homes & Gardens	01-12-05	...
01278901	Country Life	01-12-05	...
01278913	Fisherman's World	01-01-06	...
01278927	Homes & Gardens	15-08-05	...
01278944	Fashion Outlook	01-04-05	...
01278967	Fisherman's World	01-04-05	...
01278913	Offroad Vehicles	01-04-06	...
01278968			
...			

Magazine subscription database – table 2, details of subscription to company's magazines

Relational database

- search a table for all records where field has value that matches query
- combine (join) data from different tables (based on common key field)
 - generate a list of names and addresses of those customers who need to be reminded that their subscription to Fisherman's World has to be renewed next month
- update, add, delete records

Results from query on relational database

- Most SQL queries result in a table
 - column headings
 - corresponding values returned by a query.

Example query combining data from two tables could be "list names and addresses of subscribers to Fishermans World";
Would produce a result table like:

Name	Address
Smith, J.	6 High,
O'Brien,	4 Cliff,
...	

Relational database

- Major company database
 - Oracle on large Unix (or maybe IBM "mainframe")
- Personal or small business database
 - Access on PC

- Both could use **SQL** as mechanism for entering requests for searches, data joins, updates etc

SQL

- Structured Query Language
 - **SELECT** data fields wanted **FROM** database table
 - **WHERE** data in row of table matches some condition
 - **UPDATE** database table (by) **SET(ting)** field to some value
 - **WHERE** field in row with data matches some condition
- SQL statements - strategies ...
 - A simple "querying and updating system" may hide SQL, allowing user to enter requests using some form-like interface
 - User may enter SQL queries directly to get reports generated
 - *SQL statements embedded in programming language, the data retrieved are made available for use in program*

JDBC

- JDBC:
 - an API for executing SQL statements.
 - classes and interfaces written in Java
 - easy to send SQL statements to **any** relational database.
 - with Java and the JDBC API,
 - have a web page with applet that uses information obtained from a remote database.
 - all employees access databases on company intranet from mix of Macs, PCs, & Unix workstations

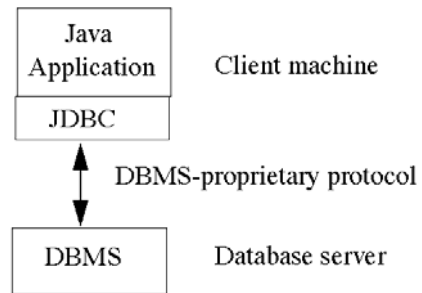
- **1. establish a connection with a database**
 - **2. send SQL statements**
 - **3. process the results:**
- ```

Connection con = DriverManager.getConnection(
 "jdbc:odbc:dbx_1", "sales", "Jx4Lq1s");
Statement stmt = con.createStatement();
ResultSet rs =
 stmt.executeQuery(
 "SELECT Name, Address FROM Table1");
while (rs.next()) {
 String s = rs.getString("Name");
 ...;
}

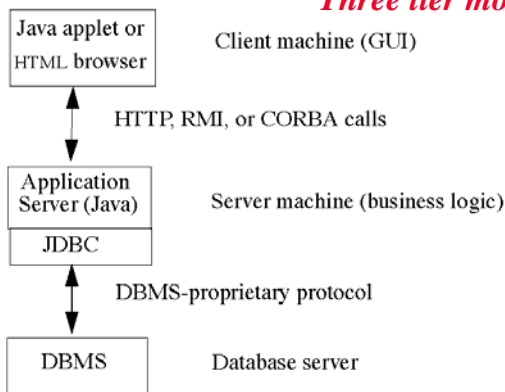
```

*Things that JDBC does*

### Two tier model



### Three tier model



### DBMS - proprietary protocol?

- Databases respond to similar SQL requests (all should support the same minimum set of SQL; many have proprietary extensions)
- But actual protocol - the set of messages exchanged between client program and database server program - vary widely.
- Something has to convert SQL queries into correct sequence of messages – this “something” is a database driver module.

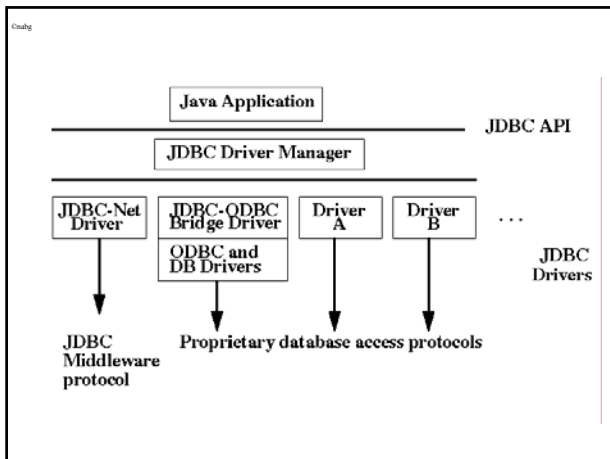
### Connect to *any* database?

- If want a program to be reasonably independent of database, need another software layer
  - something that will load appropriate database driver module
  - something that will provide a reasonably consistent interface, so that
    - can write a general piece of code with requests to send SQL queries
    - these requests converted to suit particular driver

### ODBC

- When JavaSoft started development of database facilities, there was already one fairly general package for interfacing to databases - Microsoft's ODBC (Open DataBase Connectivity package; ODBC is a standard but for a long time only Microsoft implemented it).
- JDBC can act as a Java language, object based interface to ODBC (useful if have ODBC installed and compatible drivers already implemented)
- If database developer can supply appropriate Java compatible “driver” modules, can leave out ODBC.

ODBC is a form of “C” function library, providing functions and data structures used to pass SQL requests to chosen database driver



## JDBC drivers

1. JDBC-ODBC bridge plus ODBC driver:
  - ODBC binary code, and database client code, must be loaded on each client machine
  - Appropriate for use on a corporate intranet, or for a Java server in a 3-tier system.
2. Native-API partly-Java driver:
  - converts JDBC calls into calls on the client API for DBMS.
  - requires that some binary code be loaded on each client machine.

...

## JDBC drivers

3. JDBC-Net pure Java driver:
  - translates JDBC calls into a DBMS-independent net protocol which is then translated to a DBMS protocol by a server.
4. Native-protocol pure Java driver:
  - converts JDBC calls into the network protocol used by DBMSs directly.
  - Since these protocols are proprietary, database vendors have to supply such drivers.

*These days most drivers are type-4*

## To use JDBC ...

- Need appropriate software installed on system.
- But, provided low level stuff is in place, programmers can ignore details of drivers, JDBC/ODBC bridges etc.
- Programming done using instances of various high level classes.

## JDBC classes

- Driver manager
- Connection
- Statement
  - PreparedStatement
    - CallableStatement
- ResultSet

## DriverManager

- DriverManager
  - keeps track of available drivers
  - handles establishment of connection between a database and the appropriate driver.
  - deals with login time limits, printing of log and tracing messages.

## DriverManager

- Drivers have to be loaded and registered with manager
  - possibly automatic, run-time system uses list of available drivers defined in an environment variable
  - possibly by explicitly loading code for and creating driver

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

## DriverManager

- Main function

```
public static synchronized Connection
getConnection(String url) throws SQLException
public static synchronized Connection getConnection(
 String url, String user, String password)
 throws SQLException
```

Attempt to establish a connection to the given database URL. (the appropriate driver from the set of registered JDBC drivers is identified automatically from the "database URL")

url - a database url of the form jdbc:subprotocol:subname

user - the database user

password - the user's password *Can be encoded in URL*

## "Database URLs"

- A way of identifying a database so that the right driver will recognize it and establish a connection with it.
  - first part "jdbc"
  - second part a subprotocol associated with specific driver
  - rest of a JDBC URL defines the datasource
    - name of local database or internet style machine.domain name and port
    - possibly parameters specific to particular form of database connection
    - optionally including login name and user password

## Connection

- A Connection represents a session with a specific database.
  - SQL statements are executed and results are returned.
  - transaction aspects handled (commit, rollback etc)
  - can get information from database describing
    - tables,
    - SQL grammar,
    - stored procedures, *"Metadata"*
    - capabilities of connection,

## Opening a connection

- DriverManager.getConnection.
  - takes a string containing a URL
  - checks each driver (in the list of registered drivers) until finds one that can connect to the database specified by URL

```
String url = "jdbc:odbc:csci";
Connection con =
 DriverManager.getConnection(url,
 "jg001", "t8hagq0");
```

## Sending SQL statements

- Once a Connection made, it is used to pass SQL statements to its underlying database.
  - no restrictions on SQL statements sent
  - but will get exceptions if database cannot handle SQL used in statement
- Connection creates objects that "contain SQL code" using methods
  - createStatement
  - preparedStatement
  - prepareCall

## Statements

- Statement
  - for sending simple SQL statements.
- PreparedStatement
  - for SQL statements that take one or more parameters as input arguments (IN parameters).
  - potentially more efficient as pre-compiled and stored for further use.
- CallableStatement
  - used to execute SQL stored procedures

## ResultSet

- ResultSet
  - contains the rows satisfying the conditions in an SQL statement,
  - provides access to the data in those rows (methods allow access to the various columns of the current row)
  - bit like an Iterator (Enumeration),
    - next() method to move to the next row of the ResultSet,

```
Connection dbx= DriverManager.getConnection(
 db_url, myuserid, mypassword);
Statement stmt = dbx.createStatement();
...
String query = "SELECT Name, Address FROM
CustomerTable WHERE ...";

ResultSet results =
 stmt.executeQuery(query);
while(results.next()) {
 String c_name =
 results.getString("Name");
 ...
}
```

## JDBC & Applets ?

- Some problems
  - most environments don't allow Applets to execute native code,
    - so can't use a JDBC-ODBC driver, or Native API - part Java;
  - Applets only supposed to connect to machine from which downloaded, but most organizations will not put database and web pages on same machine
    - so pretty much restricted to "three-tier" model

## JDBC example

## Bank customer table search

- Trivial example
  - define a table in sql
  - populate it
  - have a little Java program that searches through table for name of customer with specified customer-identifier-number

Crashg

## Table

```

create table customers (
 id integer,
 name varchar(32) not null,
 address varchar(64) not null,
 password varchar(8) not null,
 balance number(8,2),
 limit number(8,2),
 constraint id_cust_pkey primary key(id)
);

```

*Saved in file table.sql*

Crashg

## Some data entries

```

insert into customers values(
 1,'Dennis Smith','32 Bomba Street, Wollongong',
 'umbolt',32444.33,-5000
);
insert into customers values(
 2,'Ralph Warner','3 Alcock close, Wollongong',
 'libsta',644.52,-1000
);
insert into customers values(
 3,'Soraya Milesk','5/16 OBriens Road, Wollongong',
 'v7yme',2487.02,0
);
insert into customers values(
 4,'Fred Boshtem','11 Stafford Street, Wollongong',
 'mRkz',1456.28,0
);

```

*Saved in file data.sql*

Crashg

## Program

- Procedural style!
  - actually quite common for trivial little Java programs accessing databases
- Exception handling bit sloppy (!)
  - again quite common for trivial little Java programs accessing databases
- Program
  - Creates a connection to specified database
  - Submits a search query
  - Displays result

Crashg

## DBQuery

```

import java.sql.*;
public class DBQuery {
 private static final String userName = "HSimpson";
 private static final String userPassword = "Duh";
 ...
 private static Connection dbConnection;
 private static void connectToDatabase() { ... }
 private static void runWhoIs(int id) { ... }
 public static void main(String[] args) {
 connectToDatabase();
 runWhoIs(2);
 }
}

```

Crashg

```

import java.sql.*;
public class DBQuery {
 private static final String userName = "HSimpson";
 private static final String userPassword = "Duh";

 private static final String dbDriverName =
 "oracle.jdbc.driver.OracleDriver";
 private static final String dbURL =
 "jdbc:oracle:thin:@wraith:1521:csci";
 private static Connection dbConnection;

 ...
}

```

Crashg

```

private static void connectToDatabase() {
 try {
 Class.forName (dbDriverName);
 dbConnection = DriverManager.getConnection(
 dbURL,
 userName, userPassword);
 }
 catch(Exception e) {
 System.out.println(
 "Failed to connect to database because:");
 System.out.println(e.getMessage());
 System.exit(1);
 }
}

```

```

private static void runWhoIs(int id) {
 try {
 PreparedStatement stmt =
 dbConnection.prepareStatement(
 "select name from customers where id=?");
 stmt.setInt(1,id);
 ResultSet rset = stmt.executeQuery();
 if(rset.next()) {
 System.out.println(
 "Customer " + id + " is " +
 rset.getString("name"));
 }
 else
 System.out.println(
 "There is no customer with id " + id);
 }
 catch(Exception e) {
 System.out.println("Failed because "
 + e.getMessage());
 }
}

```

## Statement/PreparedStatement

- Statement
  - execute with string built for particular SQL query
- PreparedStatement
  - SQL contains “placeholders” for actual values to be used when run query
  - SQL is “pre-compiled” before any queries generated
  - more efficient
- (Some databases and drivers – e.g. Access – don’t support PreparedStatement; *actually, depends which version of Access you have, the latest version does support PreparedStatement*)

## Mechanics - database

- Create table and populate using tool like sqlplus

```

$ sqlplus
SQL*Plus: Release 8.1.7.0.0 - ...
Enter user-name: HSimpson
Enter password:
Connected to: ...
SQL> @table.sql;
SQL> @data.sql;

```

## Mechanics - classpath

- Have to have “.jar” file with Oracle driver
- Must add this to “CLASSPATH” used by Java program

```

prompt-javac DBQuery.java
prompt-set CLASSPATH=.;F:\OracleDrivers\ojdbc14.jar
prompt-java DBQuery
Customer 2 is Ralph Warner
prompt-

```

*Naturally, with NetBeans the mechanisms for adding .jar file to classpath are different*

For future reference:

## DriverManager etc “deprecated”

- Sun has a new set of interfaces for getting connections to databases – DataSource etc. Sun wants these to be used rather than older DriverManager.
- Reason:
  - DriverManager etc require program to have built in constants for things like URL of database and the name of the driver class
  - This makes it harder to switch database being used (must be able to recompile program)

## Datasources

- Datasources
  - View them as little data structures that a program can load at runtime
  - These structures contain all the information needed to connect to database
  - Relatively easy to substitute different data structure and so change database used.
  - Program simply has a symbolic name for the datasource structure that it wants to use

## Datasources

- But where do these data structures get loaded from?  
That symbolic name in the program – how does it get mapped to a record with the right datasource structure
- Really need supporting infrastructure – a directory service where can lookup symbolic name and find resource.
- JNDI – naming and directory services required.

## RMI

Go to sleep now – no questions on remaining topics in exam



## RMI

### Java Remote Method Invocation – Distributed Computing for Java

- Idea
  - distribute the objects involved in some application over many networked machines, these objects are located on most appropriate machine
  - distributed nature is (largely) hidden, program treats objects as local (if really a remote object, an apparent invocation of a method will result in interaction across network)

## RMI and others ...

- Ways of communicating “client to server”
  - ports & sockets
    - not hard ; Java client can communicate with servers written in other languages; *BUT - programmer must devise an application specific protocol defining how data are sent*
  - JDBC
    - fine if “server” is a database and communication limited to database queries
  - RMI
    - object oriented communication; Java to Java; no need to invent protocol, use standard mechanisms for passing arguments for method calls

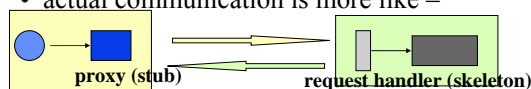
## RMI model

- One of the objects in your program needs to ask another for the date –



- but this second object is actually on another machine (it is a “remote” object)

- actual communication is more like –



## Low-level communication in RMI

- The stub and skeleton objects communicate using sockets and TCP/IP messages.
- The stub will compose a message that:
  - identifies which method is being called
  - provides the arguments, (marshalling), any objects passed as arguments in the method call are serialized
- The skeleton parses the message (creating objects if needed) and invokes the actual method of the real object.
- The skeleton “marshals” details of the result of the method call and sends back a (TCP/IP) reply.

## RMI model

- Your programs would have
  - a Calendar interface that **implements Remote** and provides functions like getDate()
- Your “client” would
  - apparently use an object that implements this Calendar interface, treating that object just like an ordinary object with method calls like getDate() and setDate(*new\_date*)
  - before using such an object, the client would have to establish the actual connection to the remote calendar implementation

## RMI model

- Your “server” program would
  - have the implementation for a class with the remote Calendar interface
- On machine with your server, you would need a registry (a server process, listening at some standard port, for queries asking about objects that provide remote access).
- Your “server” program implementing the remote calendar would have to be registered.

## RMI model

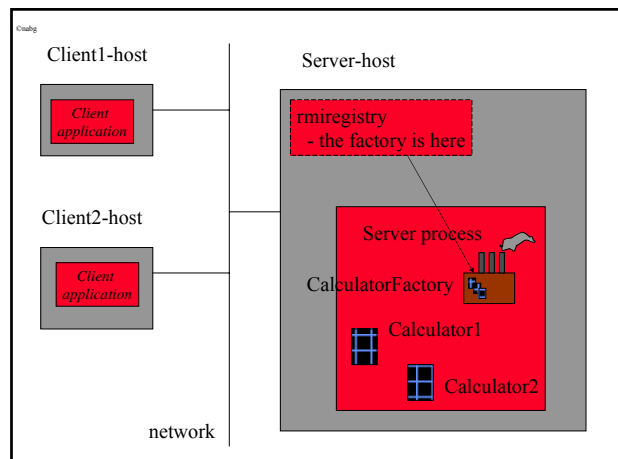
- The code for the “stub” and “skeleton” are prepared for you by the “**rmic**” tool
  - after you compile your code implementing the remote calendar, you run rmic – this produces both a xxx.Stub.class file and a xxx.Skel.class file.
- You have to get the xxx.Stub.class file onto your client machine
  - in simple cases, can just copy the file
  - more sophisticated use allows dynamic downloading of Stub.class files from a server.

*Later versions of RMI manage without specific “skeleton” classes  
Still later version omits use of “rmic” tool – stub generation is done automatically  
when stubs requested by client..*

## RMI example

## Calculator, CalculatorFactory

- Service:
  - doing calculations for clients
    - (calculations in this case are just things that can be done with a four-function pocket calculator – but you get the idea)
- Client-server behaviour
  - client contacts server process “Please issue me with a calculator”
  - Server returns a reference to a calculator (an object existing in the server process)
  - Client send calculation requests to his/her calculator



## Processes involved ...

- Client process(es)
- Server process hosting “factory” and some number of “calculator” objects
- “**rmiregistry**” – a naming service
  - bad to hardwire code with machine IP addresses and process port numbers
  - sometimes problematic to reserve port numbers for particular services
  - So idea is to have just one “well known ‘Naming’ service” (this is on fixed machine and at fixed port), all other services register when they start up
  - Client finds actual service id from Naming service

## Services

- Calculator factory
  - create a calculator when asked
- Calculator
  - do arithmetic when asked
- Services defined by interfaces

## CalculatorFactory

```
public interface CalculatorFactory extends
 java.rmi.Remote {
 public Calculator newCalculator() throws
 java.rmi.RemoteException;
}
```

## Calculator

```
public interface Calculator extends
 java.rmi.Remote {
 public int plus(int x) throws
 java.rmi.RemoteException ;
 public int minus(int x) throws
 java.rmi.RemoteException;
 public int times(int x) throws
 java.rmi.RemoteException ;
 public int divide(int x) throws
 java.rmi.RemoteException ;
 public void clear() throws
 java.rmi.RemoteException;
}
```

## Remote

- Interfaces of services “extend” **Remote**
- Remote interface
  - defines no operations!
  - defines no constants!
- Not an interface, another Java cheat,  
Remote is a modifier tag that changes how compiler handles code generation involving objects that are instances of classes implementing Remote

## RemoteException

- All operations of Remote objects defined as possibly throwing RemoteException
  - Network problems
  - Data “marshalling” problems
  - Remote server crashed
  - Remote server running but referenced server object no longer present
  - ...

## Argument and result types

- Arguments and results
  - simple types
  - anything that is “Serializable”
  - Remote types
- Pass by value
  - new object created in receiver process
  - value of a “Remote” object is an object-id (incorporates IP address, port number, and other data) – Remote tag used to flag those objects for which compiler is to generate code to deal with such object-ids [they get packaged into “stub” objects]

## Implementation

- **File CalculatorFactoryImpl.java:**

```
import java.rmi.*;
import java.rmi.server.*;
public class CalculatorFactoryImpl extends
 UnicastRemoteObject implements CalculatorFactory {
 public Calculator newCalculator()
 throws RemoteException {
 ...
 }
 public CalculatorFactoryImpl()
 throws RemoteException {
 ...
 }
}
```

## UnicastRemoteObject

- Server side object has to “attach itself” to run-time code (from java.rmi packages) that links it to the communications network and which gives it an identity that can be published to clients.
- UnicastRemoteObject class provides functions needed to such linkage to run-time.
  - either server class extends UnicastRemoteObject
    - constructor will do work necessary to link to run-time
  - or (as illustrated for Calculator implementation) the methods of UnicastRemoteObject are invoked explicitly

```
public Calculator newCalculator() throws RemoteException
{
 CalculatorImpl ci = new CalculatorImpl();
 UnicastRemoteObject.exportObject(ci);
 return ci;
}

CalculatorFactoryImpl() throws RemoteException {
 try {
 Naming.rebind(
 "rmi://localhost :13456/CalculationsAnon", this);
 }
 catch (Exception e) {
 System.out.println(e); System.exit(1);
 }
}
```

## CalculatorFactoryImpl

- Constructor invokes base class constructor (this can throw a RemoteException)
  - You can specify an explicit port that server will use; this only for special applications (allows you to avoid use of registry if you have some other “well known port”)
- Binds:
  - adds name to local rmi registry
  - “localhost”
  - “CalculationsAnon”
  - port number

## CalculatorFactoryImpl

- newCalculator() :

```
public Calculator newCalculator() throws RemoteException {
 CalculatorImpl ci = new CalculatorImpl();
 UnicastRemoteObject.exportObject(ci);
 return ci;
}
```

  - create a CalculatorImpl object (class CalculatorImpl doesn't extend UnicastRemoteObject, so instance not automatically registered with rmi run-time libraries)
  - explicitly “export” the new object
  - “return” it (because “ci” is an “exported” object, run-time support libraries know to return a proxy object rather than a serialized version of it)

## CalculatorImpl

- File CalculatorImpl.java:

```
import java.rmi.*;
import java.rmi.server.*;
public class CalculatorImpl implements Calculator {
 public int plus(int x) throws RemoteException { ... }
 public int minus(int x) throws RemoteException { ... }
 public int times(int x) throws RemoteException { ... }
 public int divide(int x) throws RemoteException { ... }
 public void clear() throws RemoteException { _val = 0; }
 public CalculatorImpl() { _val = 0; }
 private int _val;
}
```

```
import java.rmi.*;
import java.rmi.server.*;
public class CalculatorImpl implements Calculator {
 public int plus(int x) throws java.rmi.RemoteException
 {
 _val += x;
 return _val;
 }
 // Other methods left as exercise for the reader
}
```

## Next step:

*Program embodying overall server process*

```
public class Server {
 public static void main(String[] args) {
 try {
 new CalculatorFactoryImpl();
 }
 catch(Exception e) { System.out.println(e); }
 }
}
```

## Server

- Program seems to stop after creating CalculatorFactory!
- Actually, java.rmi. library code
  - creates a new “listener” port/socket combination
  - creates a thread to run accepts at this socket
  - it is this thread that keeps the server program running.

## Next step: client program using server

```
import java.io.*;
import java.rmi.*;
public class CalculatorClient {
 public static void main(String[] args) {
 try {
 CalculatorFactory theWorks =
 (CalculatorFactory) Naming.lookup(
 "rmi://banshee.cs.uow.edu.au:13456/CalculationsAnon"
);
 Calculator myCalculator = theWorks.newCalculator();
 ...
 }
 catch(Exception e) {System.out.println(e);}
 }
}
```

```
public class CalculatorClient {
 public static void main(String[] args) {
 try {
 CalculatorFactory theWorks =
 (CalculatorFactory) Naming.lookup(
 "rmi://banshee.cs.uow.edu.au:13456/CalculationsAnon");
 Calculator myCalculator = theWorks.newCalculator();
 System.out.println("I have a calculator!");
 int val = 0;
 myCalculator.clear();
 BufferedReader bin = new BufferedReader(
 new InputStreamReader(System.in));
 for(;;) { ... }
 }
 catch(Exception e) { System.out.println(e); }
 }
}
```

```

for(;;) {
 System.out.println(" " + val + " :");
 System.out.print("Command>");
 String s = (bin.readLine()).trim();
 if(s.equals("+")) {
 int arg = 0;
 s = (bin.readLine()).trim();
 arg = Integer.parseInt(s);
 val = myCalculator.plus(arg);
 }
 else
 ...
 else if(s.equals("q"))break;
}

```

## Client gets references to server objects

- Naming lookup to get reference to factory:

```

CalculatorFactory theWorks =
 (CalculatorFactory) Naming.lookup(
 "rmi://banshee.cs.uow.edu.au:13456/CalculationsAnon"
);

```
- Use factory to get reference to own calculator

```

Calculator myCalculator = theWorks.newCalculator();

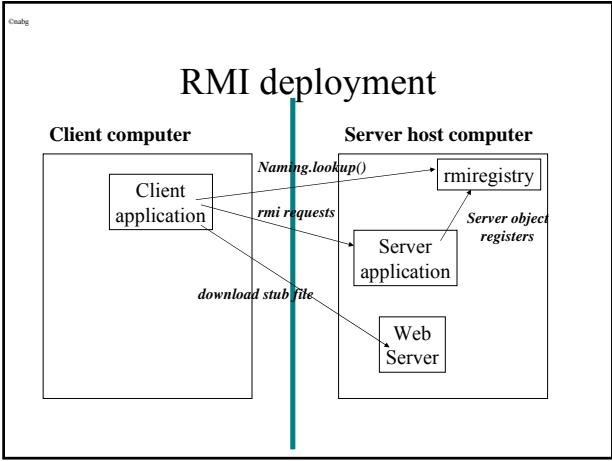
```

## Preparation

- Compile all source files.
- Run "rmic" (stub generator)
  - rmic -v1.2 CalculatorFactoryImpl
  - rmic -v1.2 CalculatorImpl
  - v1.2 option avoids generation of old style server side "skel" adapter classes
- Files: Calculator.class, CalculatorFactory.class, CalculatorImpl.class, CalculatorImpl\_Stub.class, CalculatorFactoryImpl.class, CalculatorFactoryImp\_Stub.class, Server.class, CalculatorClient.class

## Deployment

- Rules for correct deployment are rather complex!
- Correct deployment requires that you have a web-server running from which client downloads "stub" class!



## Naïve deployment

- NOT the right way.
- Following example just explains how to get things to run for those who want to try playing with rmi

## Naïve deployment

- **Simply to get a working demo** (this is not how it should be done for real)
  - Start four separate terminal sessions
  - Arrange so that each terminal session is using same directory where all .class files are located
  - Start private rmiregistry, specifying port (`$ rmiregistry 13456`)
  - Start server process (`$ java Server`)
  - Start first client (`$ java CalculatorClient`)
  - Start second client (`$ java CalculatorClient`)
- Use clients and demonstrate that non-interfering
- *(Try divide by zero, and see exception returning to client and not disrupting server process)*

## IDL

### Interface Definition Language

## IDL

- RMI intended for Java-to-Java communication.
- In real world, likely to need to communicate with object-oriented systems written in C++, or Smalltalk, or possibly other languages.
- Would still like
  - “objects” to be transferred, or be represented by proxies as appropriate.
  - method call style coding in client (rather than explicit use of ports-sockets etc)

## IDL – context

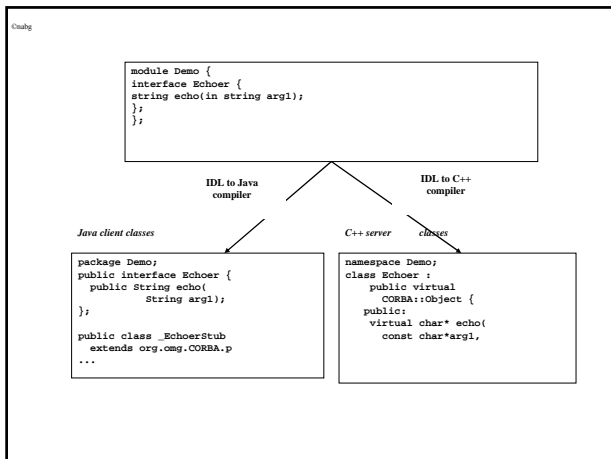
- “Object Management Group” (OMG)
  - consortium of ≈600 companies,
  - founded ≈1989
  - aim:
    - provide a general language independent mechanism for object based communications across the networks
  - before Java
    - already had mechanisms established, primarily for C++ but also dealing with Smalltalk, Objective C, C (faking objects), “Object Cobol”, ...

## OMG

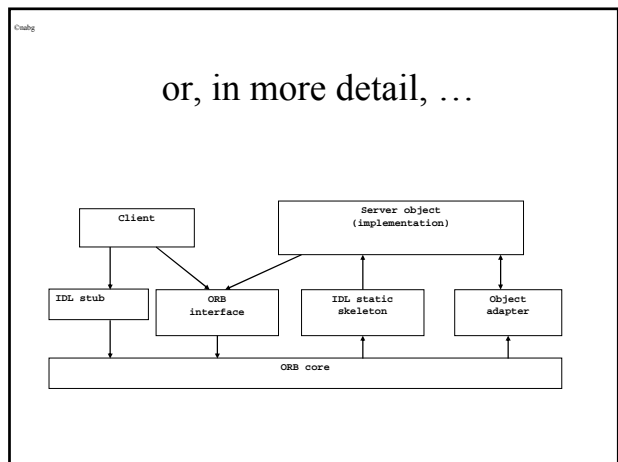
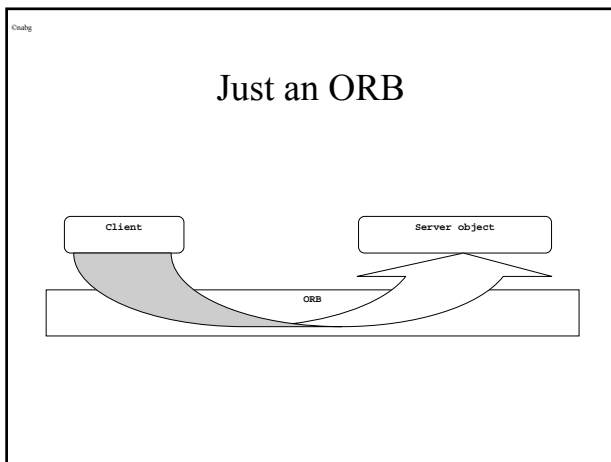
- Common Object Request Broker Architecture (**CORBA**)
  - client requiring remote services contacts an ORB, ORB identifies object that can provide needed service and forwards request
- Common Object Services Specification
  - standards for operations like creating, replicating, or moving, naming objects; handling transactions; ...

## CORBA

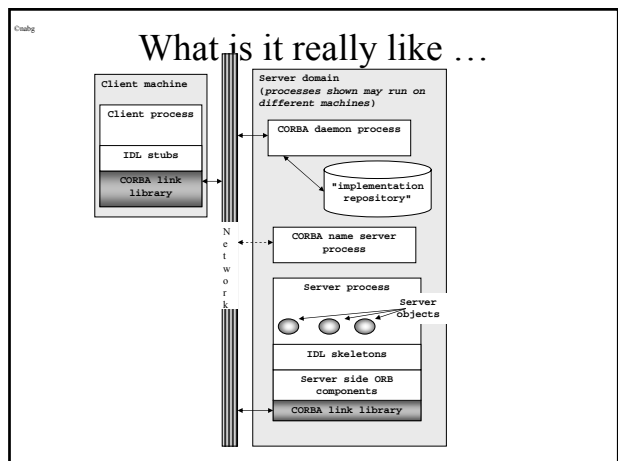
- Systems built around CORBA are intended to be independent of implementation language used in clients and servers.
- Objects that are to be in common, are required to implement defined interfaces.
- These interfaces are written in “**Interface Definition Language**” (IDL)
  - IDL class declaration says what object can do
  - different language implementations are permitted – so have an “IDL” to “host language” compiler



- ## CORBA system
- IDL compilers
    - IDL->Java
    - IDL->C++
    - others?
  - Run-time libraries linked with code
    - "Object Request Broker"
      - Fairly simple on client (similar to Java RMI)
      - Quite sophisticated on server
  - Support services
    - NamingService
    - Trader, Transactions, Events, ...



- ## What's different from RMI?
- Visible infrastructure
    - CORBA defines the interfaces of some of the infrastructure components such as the ORB
    - Standardizes how things work across differing implementations
  - Object adapter
    - OA component to allow for more control on object lifetimes



## On a CORBA client

- Client code
- Instances of stub classes generated by IDL compiler
- Client side ORB components
  - Relatively simple
  - “ORB” with a few methods like one for getting a reference to a standard service (name server, transaction server, ...)
  - Client stubs invoke operations in the run-time library

## On a server host or host group

- Server programs
  - Manually launched
  - Started automatically by CORBA daemon
- CORBA daemon process
  - can launch server processes
- Interface Repository
  - Handles queries about classes
    - Dynamic CORBA
    - Inheritance relationships in normal CORBA
- Name server

## Server side complexity

- CORBA ORB components provide an elaborate system for controlling life-cycle of server objects
  - Scalability – can have millions of objects (e.g. objects representing individual bank accounts) but contrive to use only limited storage
- More difficult to learn than RMI

## Support services

- CORBA
  - sophisticated naming service (better than rmiregistry)
  - “trader” service – find service by characteristics rather than name
  - transaction service – coordinate updating of multiple databases
  - events – publish & subscribe relations instead of simple client/server
  - ...

## CORBA: Importance?

- Mainly as an integration technology when have existing (“legacy”) systems that must be combined
- Used in telecommunications and banking
- Specialist use (real-time CORBA) in military systems

## Application servers

Servlet/JSP

EJB

Crabg

## Services

- Most applications in industry or commerce involve “services”
  - service
    - access to persistent data
    - execution of business logic
- These services used from “clients”
  - Data input and display separated from underlying business logic
  - This “Presentation” layer runs on separate machine – desktop workstation, laptop with radio-network,

Crabg

## Distributed systems

- Late 1980s-mid-1990s: two tier client server on local networks (client:presentation and business logic; database)
- Multi-tier applications are now the norm.
  - Client tier
    - Data input, results presentation
    - Commonly web-based; but can be specially written using TCP/IP communications

- Web tier
  - Intermediary to deal with HTTP clients
- Middleware tier
  - Business logic

- Database tier
  - Data persistence

Crabg

## Development?

- Client?
  - If using Web, then nothing much needed
    - Data entry forms – declarative HTML
    - A little client side Javascript
  - Otherwise, a GUI client (Java or “Winforms”) that uses TCP/IP connections to middle layer
- Database?
  - Best if just an engine for storing and retrieving data, but can put some business rules in as code run on database machine (e.g. Oracle can handle scripts in Java)
  - Design the tables
  - Write the SQL statements to store and retrieve data

*Winforms: Microsoft technology, typically Visual Basic, possibly C#*

Crabg

## Development?

- **The middle layer**
  - Write your own program – starting by using ports and sockets to pick up communications from clients?

**Not any more!**

- Build something using RMI or CORBA

**Rarely!**

Crabg

## Container & component architecture

- Most current commercial software development based on an architecture rather different from any you have met so far.
- Architecture exploits fact that most applications have numerous common aspects
  - low-level communications (layered over TCP/IP or on top of HTTP as a carrier)
  - resource management (e.g. database connections)
  - controlled access (“security”)
  - lifecycle management

Crabg

## Container & component architecture

The diagram illustrates a container-based architecture. At the top, a large grey rectangle labeled "Container" contains two colored circles: a blue circle labeled "business component A" and a red circle labeled "business component B". Below the container, there are four grey rectangular boxes arranged in a 2x2 grid. The top-left box is labeled "Communications", the top-right is "Transactions", the bottom-left is "Database access", and the bottom-right is "component management".

## Factor out the common code

- As observed with CORBA (and Microsoft DCOM) programs in mid-1990s
  - *Most applications had similar code*
    - CORBA (DCOM) library handled low-level communications (all hidden from programmer)
    - CORBA library provided functions that programmer could use to build lifecycle management, resource management, controlled access etc
    - CORBA objects created to handle business processing – *link into CORBA run-time (repetitious complex code)*
    - Programmers kept *reimplementing essentially same code*
  - Factor out the common code
    - Provide a richer framework – hides not just communications level, hides all the other messy standard parts!

## Container-component

- Container
  - code to handle the communications, lifecycle management etc, etc
  - API defining services that container provides to component ( ... “give me the data from the client”, ... “send this response to client”, ... “give me a connection to database”, ... )
- Component
  - Subclass of framework defined base class (like RMI server being subclass of UnicastRemoteObject)
  - Framework base class supplies functionality needed to tie into container
  - Implements just the business code.

## Containers support specific application types

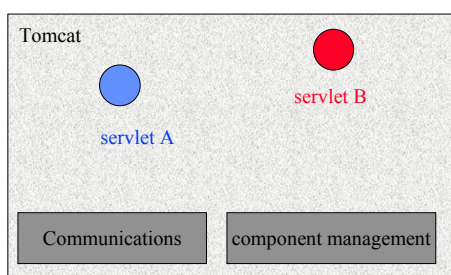
- Frameworks for particular types of application
  - “The Web Application”
    - Get data from a HTML form
    - Simple database access
    - Generate response
  - “The Enterprise Application”
    - Access and update data in multiple datastores (dealing with multiple concurrent activities affecting shared data)
    - Carry out “workflow” operations that involve maintaining temporary state
    - e.g. doing your on-line banking!

## Servlet container & servlets

- “Servlet”
  - An object that handles processing of data from a particular WWW form
  - Defined base class HttpServlet
    - Programmer implements doGet/doPost functions
- Servlet container – e.g. Tomcat
  - Manages servlets
  - Provides access controls
  - Supports sharing of data among related servlets
  - ...

You get to work with servlets in CSCI399

## Tomcat



## JSP

- Java Server Page
  - a technology built on top of servlets
  - servlets good for organizing the processing of web requests, but rather poor way of generating the response page (too much code, makes it difficult to make page pretty or to change page appearance in later version of application)
  - JSPs focus on presentation of data
  - Best used together
    - servlet organizes data processing, creates “beans” (struct like things) with results, passes result beans to JSP
    - JSP creates pretty response page using HTML and data in beans

## Enterprise Java Beans Container

- Enterprise Java Beans
  - “Entity” beans – used to access persistent data
  - “Session” beans
    - Business rules
- EJB Container services
  - Lifecycle management of “beans”
  - Pooled database connections
  - Transaction control (really this means a mechanism for locking resources shared by many clients so that their operations cannot interfere and produce erroneous results)
  - ...

*You get to talk about EJBs and IBM's WebSphere container in some of eBusiness subjects*

## Messaging, WebServices

## Messaging

- **Client-server relations are typically synchronous**
  - Client submits request, waits for response
- **Sometimes more appropriate to dispatch a request for work to be done by a service, receive a message later when work done**
- Messaging systems work with intermediary processes that queue up those requests for work
- APIs define how
  - client can compose work requests,
  - client can dispatch requests to queue
  - server can select work from queue
  - server can send response (if needed)
  - client can either be interrupted by response, or can ask to collect response at its convenience

## WebServices

- Nothing really new
- Just an alternative way of defining service interfaces (and some of CORBA style support services like transactions), plus another set of communications protocols
- But WebService definitions are technology neutral
  - Same service interface can be used to generate components for EJB/CORBA/.Net
  - Overall system can use some components deployed in a .Net container, others in an EJB container, others in a CORBA system – all will work together using WebService communications protocols

## WebServices – another integration technology

- CORBA, Microsoft DCOM, RMI all provide ways of defining “services”
  - Interface
    - Lists operations (functions)
    - Defines data types
  - Transport protocol for messages invoking services
- These service interface definitions used to generate client-stubs (and server skeletons where needed)
- But they are not interoperable (require clumsy bridging systems to get CORBA to talk to DCOM)

*Microsoft: COM, DCOM, COM+, and now .Net*