Netværksprogrammering

Plan

- Socket-baseret kommunikation
- Fjernmetodekald (RMI)
  - Designmønsteret Proxy
- Databasetilgang (JDBC)

Distribuerede beregninger

Dagens beregningsmiljøer er

distribuerede: beregningerne forgår på flere værtsmaskiner

heterogene: værtsmaskinerne er forskellige

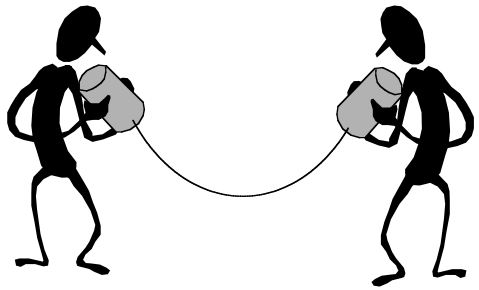
Kommunikation

Java indeholder to mekanismer til distribuerede beregninger:

(1) *Socket-baseret kommunikation* (`java.net`)
En "socket" er et endepunkt for en tovejs-forbindelse imellem to parter.

(2) *Fjernmetodekald* (RMI) (`java.rmi`)
RMI tillader manipulation af objekter på fjerne maskiner, som om objekterne var lokale.

Socket-baseret kommunikation



Sockets er endepunkterne for en tovejs-forbindelse imellem to værtsmaskiner.

Der er to slags sockets: *server sockets* og *client sockets*.

En *server socket* venter på klienters anmodning om en forbindelse.

En *client socket* (også blot kaldet *socket*) bruges til både at sende og modtage data.

Porte

En server socket lytter på en bestemt *port*.

En port er angivet ved et positivt heltal mindre end eller lig med 65565.

Portnummeret er nødvendigt for at kunne skelne imellem servere (server-applikationer), der kører på samme værtsmaskine.

De første 1024 porte er reserveret til specifikke protokoller, f.eks. 21 til FTP, 23 til Telnet, 25 til e-mail og 80 til HTTP.

Server sockets

En server socket er en instans af klassen `ServerSocket` og skabes ved en af følgende to konstruktører:

```
ServerSocket(int port)
ServerSocket(int port, int backlog)
```

`port`: portnummeret

`backlog`: den maksimale længde for køen af ventende klienter (50 er standard)

Kun applikationer kan skabe `ServerSocket`-objekter.

Metoder i `ServerSocket`

`Socket accept()`

Vent på en anmodning om en forbindelse fra en klient. Den tråd, der udfører kaldet, blokeres, indtil der modtages en anmodning om forbindelse. Når det sker, returnerer `accept` med et `Socket`-objekt.

`void close()`

Luk denne `ServerSocket`.

Typisk brug af `ServerSocket`

```
try {
    ServerSocket s = new ServerSocket(port);
    while (true) {
        Socket incoming = s.accept();
        «Handle a client»
        incoming.close();
    }
    s.close();
} catch (IOException e) {
    «Handle exception»
}
```

Client sockets

En client socket er en instans af klassen `Socket` og kan opnås på to måder:

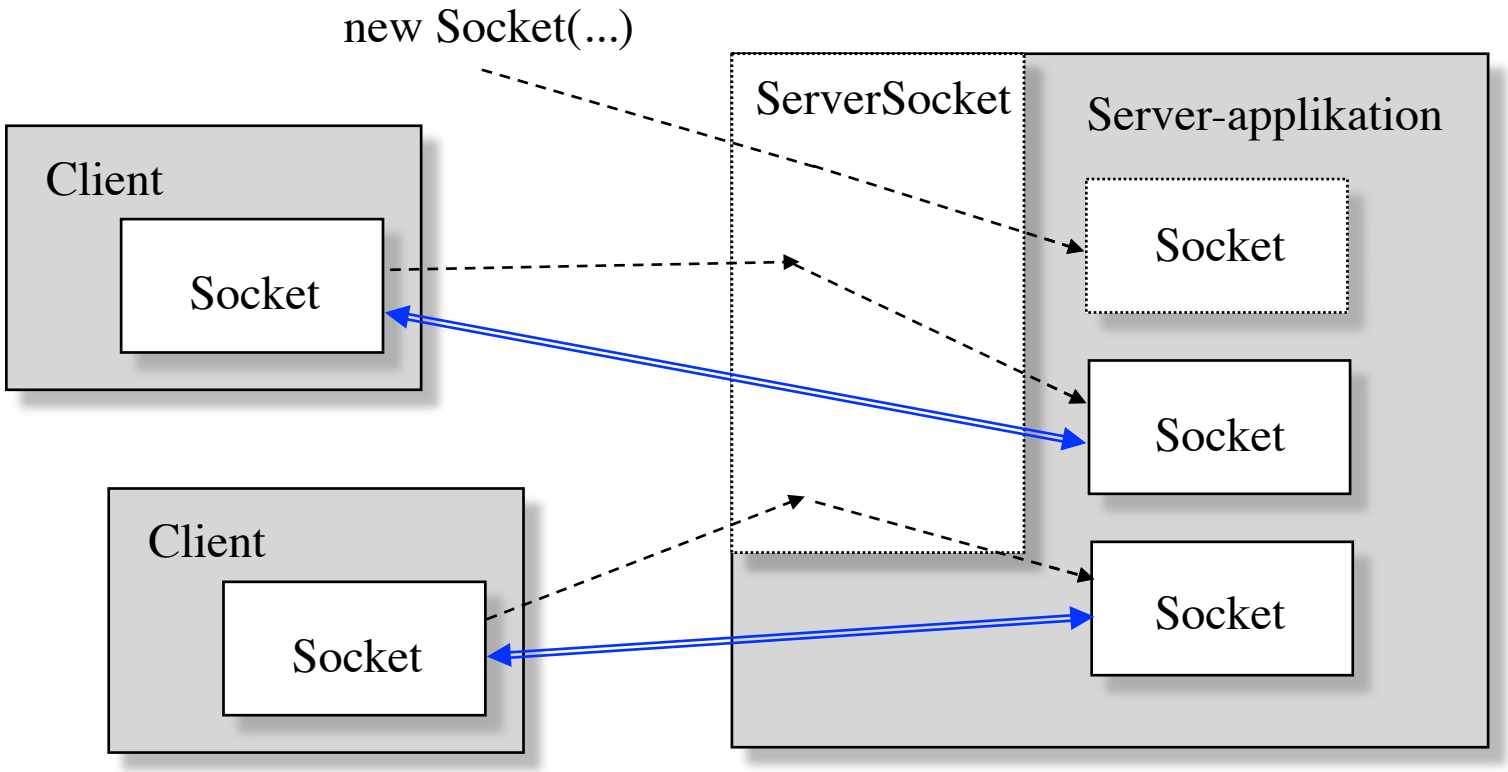
(1) På serversiden som returværdi af `accept()`.

(2) På klientsiden ved brug af konstruktøren

```
Socket(String host, int port)
```

`host`: navnet på værtsmaskinen
`port`: portnummeret

Klienters kommunikation med en server



Kommunikationen håndteres på begge sider med `Socket`-objekter.

Metoder i `Socket`

`getInputStream()`

Returnerer et `InputStream`-objekt til modtagelse af data

`getOutputStream()`

Returnerer et `OutputStream`-objekt til afsendelse af data

`close()`

Lukker forbindelsen

Typisk brug af `Socket`

```
try {
    Socket socket = new Socket(host, port);
    BufferedReader in = new BufferedReader(
        new InputStreamReader(
            socket.getInputStream()));
    PrintWriter out = new PrintWriter(
        new OutputStreamWriter(
            socket.getOutputStream()));
    «Send and receive data»
    in.close();
    out.close();
    socket.close();
} catch (IOException e) {
    «Handle exception»
}
```

Udvikling af client/server-programmer

1. Afgør, om det er fornuftigt at implementere en server og en eller flere matchende klienter
2. Design en (tekstbaseret) kommunikationsprotokol
3. Implementér serveren
4. Afprøv serveren med et telnetprogram
5. Programmér og afprøv en klient

En simpel ekko-server


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

public class EchoServer {
    public static void main(String[] args) {
        try {
            ServerSocket s = new ServerSocket(8008);
            while (true) {
                Socket incoming = s.accept();
                BufferedReader in = new BufferedReader(
                    new InputStreamReader(
                        incoming.getInputStream()));
                PrintWriter out = new PrintWriter(
                    new OutputStreamWriter(
                        incoming.getOutputStream()));
```

fortsættes

```
                out.println("Hello! This is the Java EchoServer.");
                out.println("Enter BYE to exit.");
                out.flush();
                while (true) {
                    String str = in.readLine();
                    if (str == null)
                        break; // client closed connection
                    out.println("Echo: " + str);
                    out.flush();
                    if (str.trim().equals("BYE"))
                        break;
                }
                in.close();
                out.close();
                incoming.close();
            }
        } catch (Exception e) {}
    }
}
```

Kørsel med et telnetprogram som klient

```
venus% telnet saturn 8008
Trying 140.192.34.63 ...
Connected to saturn.
Escape character is '^]'.
Hello! This is the Java EchoServer.
Enter BYE to exit.
Hi, this is from venus
Echo: Hi, this is from venus
BYE
Echo: BYE
Connection closed by foreign host.
```

Telnet: Et terminalemuleringsprogram til TCP/IP-netværk såsom Internettet

En simpel Java-klient til ekko-serveren

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

public class EchoClient {
    public static void main(String[] args) {
        try {
            String host =
                args.length > 0 ? args[0] : "localhost";
            Socket socket = new Socket(host, 8008);
            BufferedReader in = new BufferedReader(
                new InputStreamReader(
                    socket.getInputStream()));
            PrintWriter out = new PrintWriter(
                new OutputStreamWriter(
                    socket.getOutputStream()));
```

fortsættes

```
                // send data to the server
                for (int i = 1; i <= 10; i++) {
                    System.out.println("Sending: line " + i);
                    out.println("line " + i);
                    out.flush();
                }
                out.println("BYE");
                out.flush();

                // receive data from the server
                while (true) {
                    String str = in.readLine();
                    if (str == null)
                        break;
                    System.out.println(str);
                }
                in.close();
                out.close();
                socket.close();
            } catch (Exception e) {}
        }
    }
```

Kørsel med Java-klienten

```
venus% java EchoClient saturn
Sending: line 1
Sending: line 2
...
Sending: line 10
Hello! This is Java EchoServer.
Enter BYE to exit.
Echo: line 1
Echo: line 2
...
Echo: line 10
Echo: BYE
```

En ekko-server, der servicerer flere klienter samtidigt

Benyt en tråd for hver af klienterne.

```
public class MultiEchoServer {
    public static void main(String[] args) {
        try {
            ServerSocket s = new ServerSocket(8009);
            while (true) {
                Socket incoming = s.accept();
                new ClientHandler(incoming).start();
            }
        } catch (Exception e) {}
    }
}
```

Klassen `ClientHandler`

```
public class ClientHandler extends Thread {
    protected Socket incoming;

    public ClientHandler(Socket incoming) {
        this.incoming = incoming;
    }

    public void run() {
        try {
            BufferedReader in = new BufferedReader(
                new InputStreamReader(
                    incoming.getInputStream()));
            PrintWriter out = new PrintWriter(
                new OutputStreamWriter(
                    incoming.getOutputStream()));
```

fortsættes

```
                out.println("Hello! ...");
                out.println("Enter BYE to exit.");
                out.flush();
                while (true) {
                    String str = in.readLine();
                    if (str == null)
                        break;
                    out.println("Echo: " + str);
                    out.flush();
                    if (str.trim().equals("BYE"))
                        break;
                }
                in.close();
                out.close();
                incoming.close();
            } catch (Exception e) {}
        }
    }
```

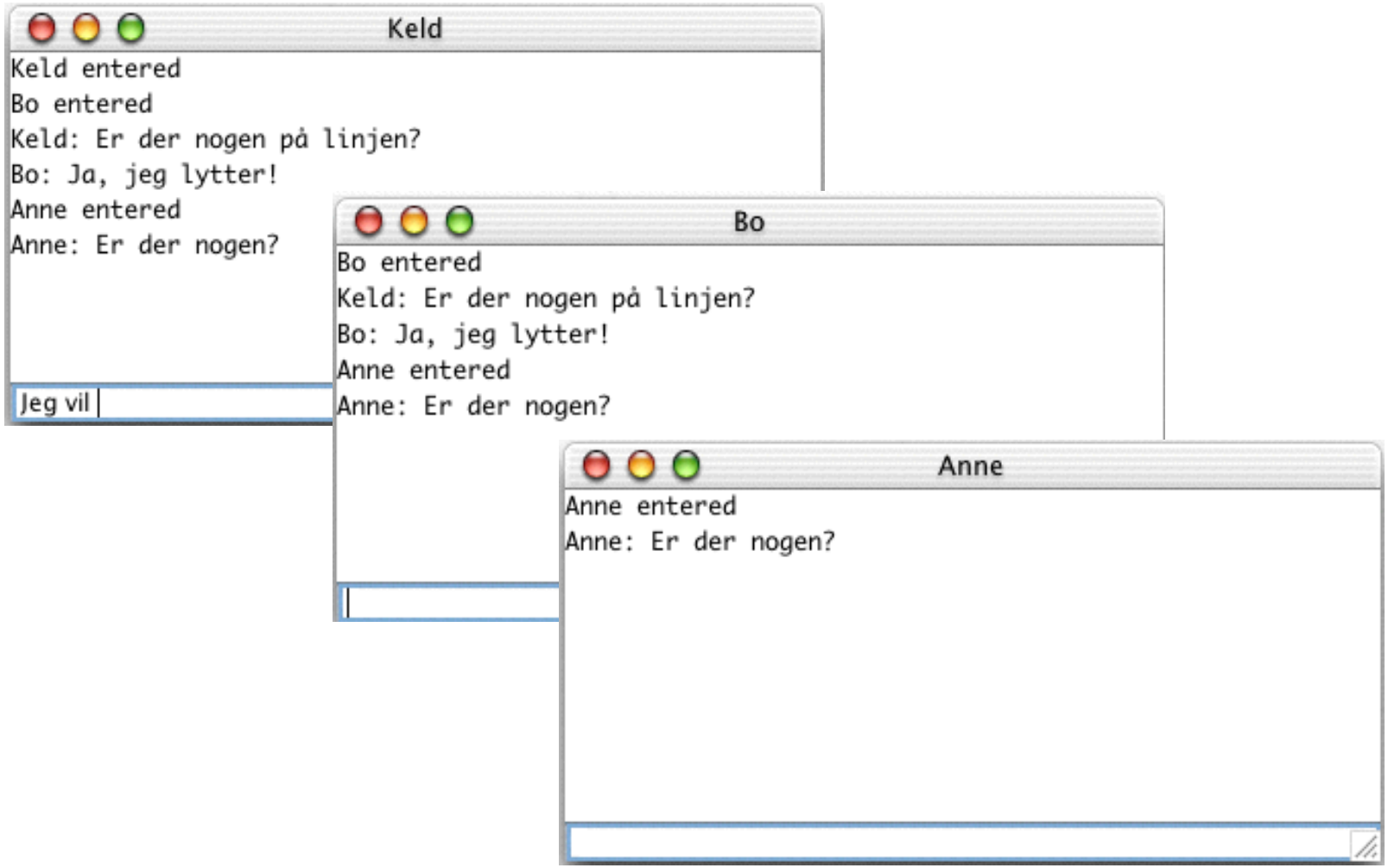
Rundsending af meddelelser til flere klienter

Udvikling af en chat-server, der

- håndterer flere klienter samtidigt
- rundsender en meddelelse, der modtages fra en klient til alle andre klienter

Der er behov for at holde rede på alle aktive klienter.

Chat


Keld
Keld entered
Bo entered
Keld: Er der nogen på linjen?
Bo: Ja, jeg lytter!
Anne entered
Anne: Er der nogen?
Jeg vil
Bo
Bo entered
Keld: Er der nogen på linjen?
Bo: Ja, jeg lytter!
Anne entered
Anne: Er der nogen?
Anne
Anne entered
Anne: Er der nogen?

ChatServer

```
public class ChatServer {
    public ChatServer(int port) throws IOException {
        ServerSocket s = new ServerSocket(port);
        while (true)
            new ChatHandler(s.accept()).start();
     }

    public static void main(String[] args)
                              throws IOException {
        if (args.length != 1)
            throw new RuntimeException(
                        "Syntax: java ChatServer <port>");
        new ChatServer(Integer.parseInt(args[0]));
    }
}
```

ChatHandler

```
public class ChatHandler extends Thread {
    Socket socket;
    DataInputStream in;
    DataOutputStream out;
    static Set<ChatHandler> handlers = new HashSet<ChatHandler>();

    public ChatHandler(Socket socket) throws IOException {
        this.socket = socket;
        in = new DataInputStream(socket.getInputStream());
        out = new DataOutputStream(socket.getOutputStream());
        handlers.add(this);
    }
```

fortsættes

```
public void run() {
    String name = "";
    try {
        name = in.readUTF();
        System.out.println("New client " + name + " from " +
                           socket.getInetAddress());
        broadcast(name + " entered");
        while(true)
            broadcast(name + ": " + in.readUTF());
    } catch (IOException e) {
        System.out.println("-- Connection to user lost.");
    } finally {
        handlers.remove(this);
        try {
            broadcast(name + " left");
            in.close();
            out.close();
            socket.close();
        } catch (IOException e) {}
    }
}
```

fortsættes

```
    static synchronized void broadcast(String message)
                                throws IOException {
        for (ChatHandler handler : handlers) {
            handler.out.writeUTF(message);
            handler.out.flush();
        }
    }
}
```

Bemærk, at metoden skal være `synchronized`.

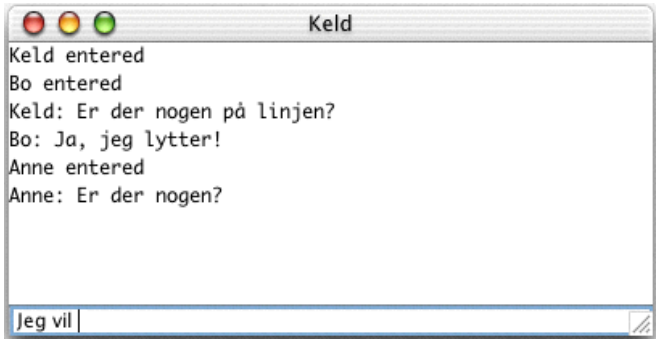
ChatClient

```
public class ChatClient {
    String name;
    Socket socket;
    DataInputStream in;
    DataOutputStream out;
    ChatFrame gui;

    public ChatClient(String name, String server, int port) {
        try {
            this.name = name;
            socket = new Socket(server, port);
            in = new DataInputStream(socket.getInputStream());
            out = new DataOutputStream(socket.getOutputStream());
            out.writeUTF(name);
            gui = new ChatFrame(this);
            while (true)
                gui.output.append(in.readUTF() + "\n");
        } catch (IOException e) {}
    }
```

fortsættes

```
    void sendTextToChat(String str) {
        try {
            out.writeUTF(str);
        } catch (IOException e) { e.printStackTrace(); }
    }

    void disconnect() {
        try {
            in.close();
            out.close();
            socket.close();
        } catch (IOException e) { e.printStackTrace(); }
    }

    public static void main(String[] args) throws IOException {
        if (args.length != 3)
            throw new RuntimeException(
                "Syntax: java ChatClient <name> <serverhost> <port>");
        new ChatClient(args[0], args[1], Integer.parseInt(args[2]);
    }
}
```

ChatFrame




```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class ChatFrame extends JFrame {
    JTextArea output = new JTextArea();
    JTextField input = new JTextField();

    public ChatFrame(final ChatClient client) {
        super(client.name);
        Container pane = getContentPane();
        pane.setLayout(new BorderLayout());
        pane.add(new JScrollPane(output), BorderLayout.CENTER);
        output.setEditable(false);
        pane.add(input, BorderLayout.SOUTH);
```

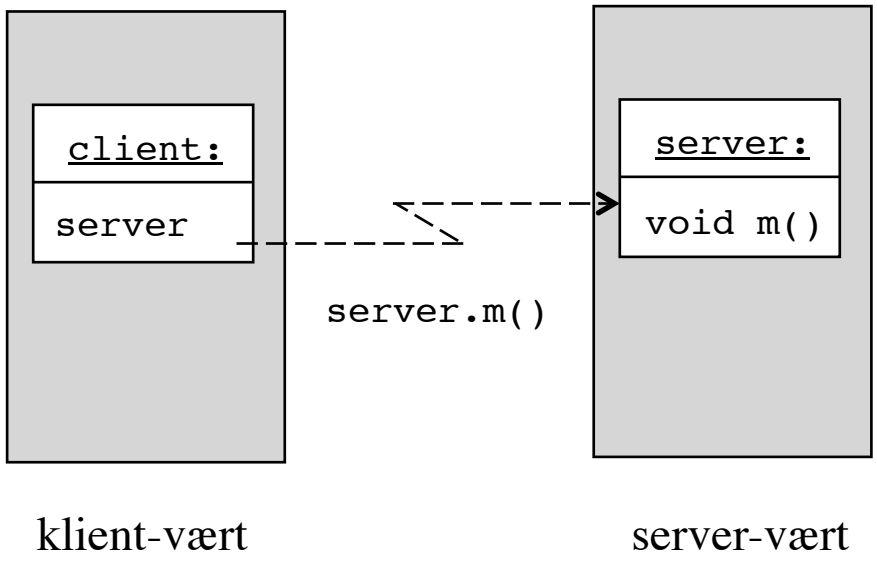
fortsættes

```
            input.addKeyListener(new KeyAdapter() {
                public void keyPressed(KeyEvent e) {
                    if (e.getKeyCode() == KeyEvent.VK_ENTER) {
                        client.sendTextToChat(input.getText());
                        input.setText("");
                    }
                }
            });
            addWindowListener(new WindowAdapter() {
                public void windowClosing(WindowEvent e) {
                    client.disconnect();
                    System.exit(0);
                }
            });
            setSize(400, 200);
            setVisible(true);
            input.requestFocus();
        }
    }
```

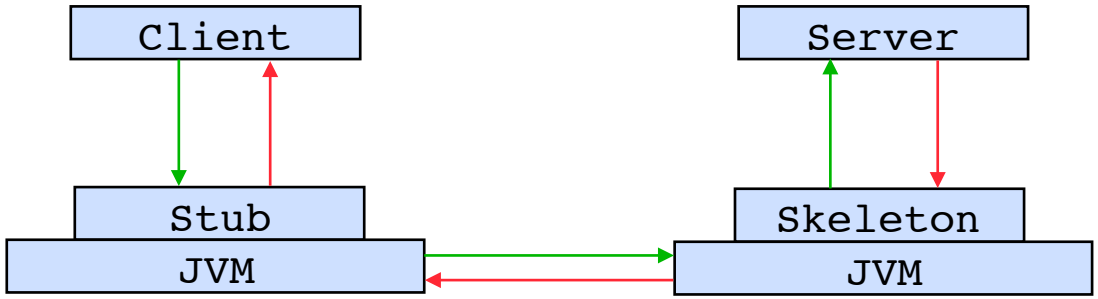
Fjernmetodekald

RMI (**R**emote **M**ethod **I**nvocation)

Objekter på fjerne værtsmaskiner kan manipuleres, som om de befandt sig på den lokale maskine.

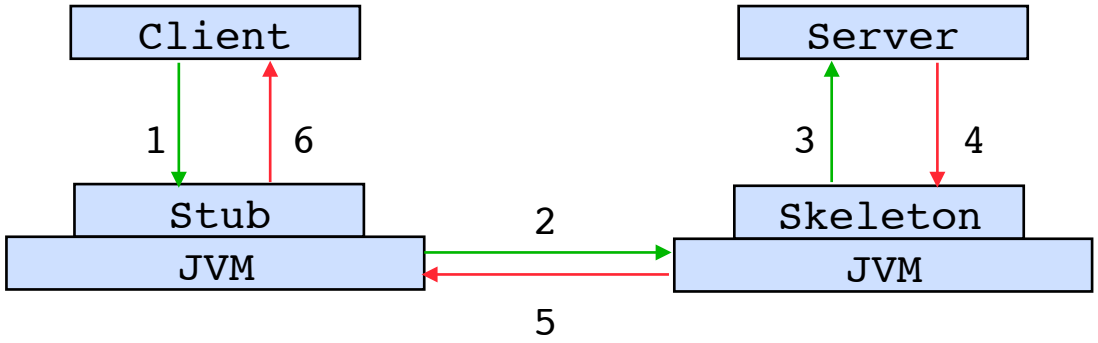


RMI-arkitekturen



Client:
Et objekt på klient-værten
Server:
Et objekt på server-værten, der servicerer klienter
Stub:
Et objekt på klient-værten, der udgør en erstatning for Server-objektet
Skeleton:
Et objekt på server-værten, der modtager forespørgsler fra stubbe og videresender dem til Server-objektet
Service contract:
En grænseflade, der specificerer, hvilke tjenester, serveren kan yde

RMI-kald



Kald af `server.m()` i klienten:

1. Den tilsvarende stubmetode, `stub.m()`, kaldes
2. Stubben serialiserer argumenter og sender argumenter og kaldinformation til skelettet
3. Skelettet deserialiserer argumenter og kaldinformation, og kalder metoden `m()` i serveren
4. Serveren udfører metoden og returnerer resultatet til skelettet
5. Skelettet serialiserer resultatet og sender det til stubben
6. Stubben deserialiserer resultatet og returnerer det til klienten

RMI-programmering

`Server`, `Client` og `Service contract` skrives af programmøren.

Stubbe og skeletter genereres af en RMI-oversætter (f.eks. `rmic`) ud fra den oversatte `Server`-klasse.

Parameteroverførsel

Ved et fjernmetodekald overføres *lokale* (nonremote) objekter ved værdi-overførsel (serialisering), medens *fjerne* (remote) objekter overføres ved reference-overførsel.

RMI-registrering

Enhver RMI-server identificeres ved en URL med protokollen `rmi`.

```
rmi://host:port/name
```

`host`: værtsnavn for RMI-kataloget (hvis udeladt: `localhost`)
`port`: portnummer for RMI-kataloget (hvis udeladt: 1099)
`name`: navnet tilknyttet serveren

Serveren registreres på server-værten i et RMI-katalog.
Denne proces kaldes *binding*:

```
Naming.bind(name, server)
```

Opslag i RMI-kataloget

En klient kan lokalisere en RMI-server ved at slå op i RMI-kataloget:

```
Remote server = Naming.lookup(url)
```

Operationer på RMI-kataloget

(statiske metoder i `Naming`)

```
static void bind(String name, Remote obj)

static void rebind(String name, Remote obj)

static void unbind(String name)

static Remote lookup(String url)

static String[] list(String url)
```

Anvendelse af RMI

1. Definer en grænseflade for Server-objektet.

```
public interface Contract extends Remote {
    public void aService(...) throws RemoteException;
    // other services
}
```

Denne udgør kontrakten imellem serveren og dens klienter.
Grænsefladen skal udvide grænsefladen `Remote`.
Hver af servicemetoderne skal erklæres, således at den kan kaste en `RemoteException`.
Argumenternes typer skal være serialiserbare.

fortsættes

2. Definer en klasse, der implementerer kontrakten.
   Klassen skal nedarve fra klassen `UnicastRemoteObject`.

```
public class ServiceProvider extends UnicastRemoteObject
                                implements Contract {
    public void aService(...) throws RemoteException {
        // implementation
    }
    // implementation of other services
}
```

fortsættes

3. Skab en instans af serveren og registrer den i RMI-kataloget.

```
Contract remoteObj = new ServiceProvider(...);
Naming.rebind(name, remoteObj);
```

4. Generer stub- og skelet-klasserne ved hjælp af RMI-oversætteren.

fortsættes

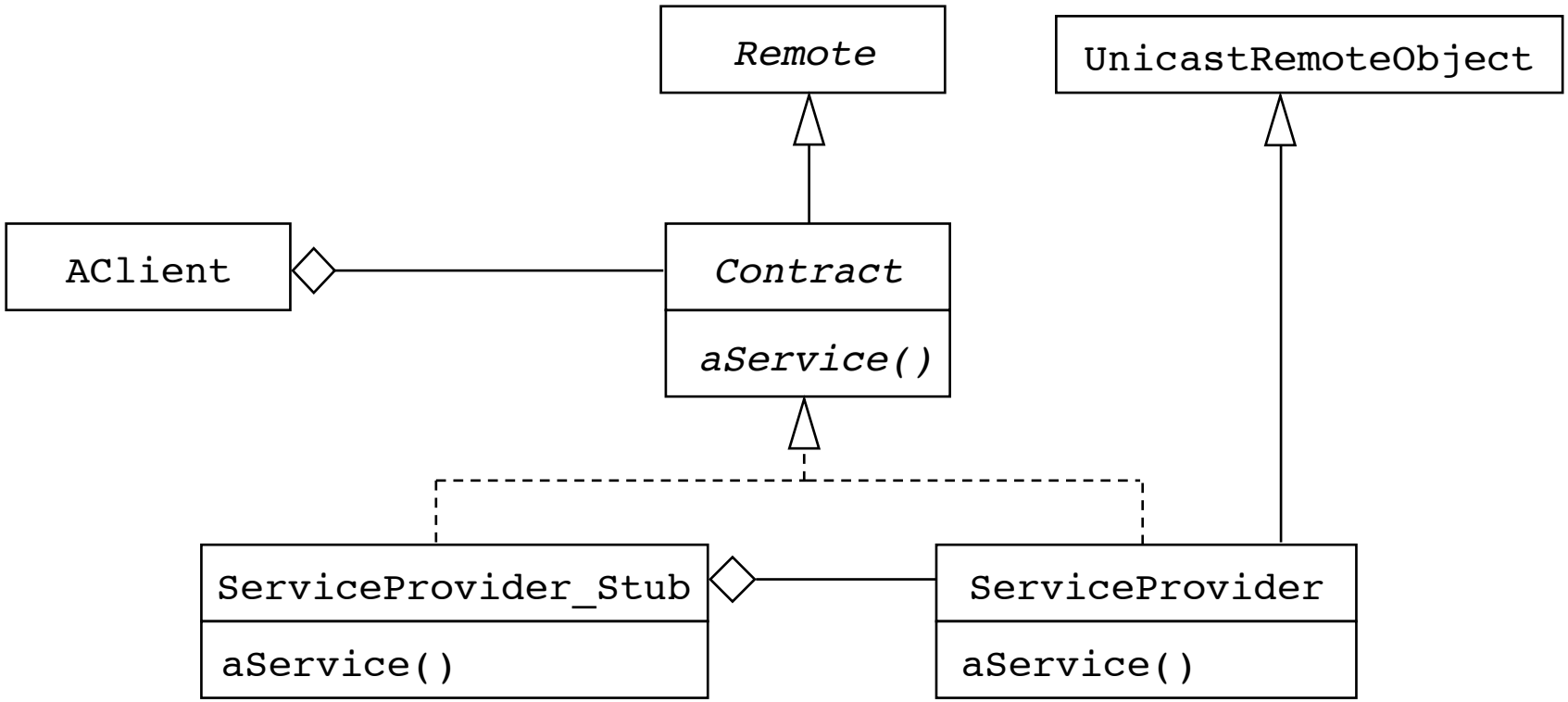
5. Programmér en klient, der bruger de tjenester, som serveren udbyder.

   For at bruge et fjernt objekt, må klienten først lokalisere objektet ved hjælp af RMI-kataloget.

```
Contract serverObj = (Contract) Naming.lookup(url);
//...
serverObj.aService(...);
//...
```

fortsættes

Struktur af RMI-applikationer



Designmønsteret Proxy

Kategori:

Strukturelt designmønster

Hensigt:

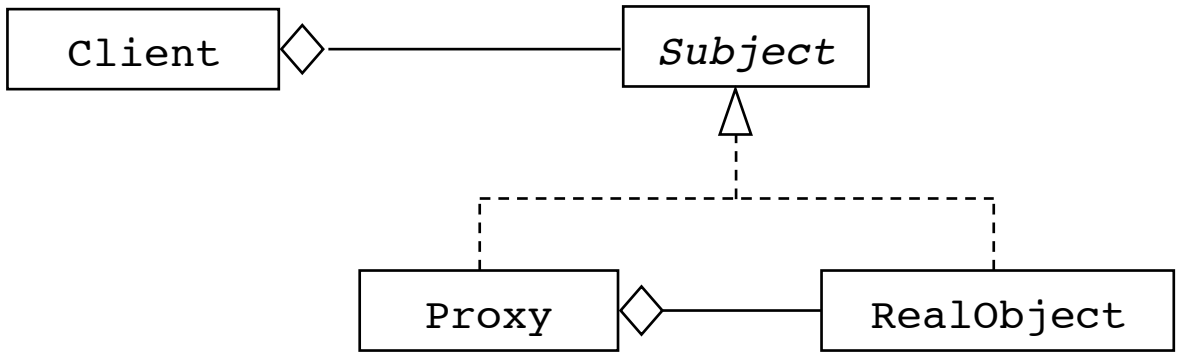
At tilbyde en erstatning for et andet objekt.

Anvendelse:

- Når der er brug for en mere raffineret reference til et objekt end en simpel reference

Designmønsteret Proxy
(fortsat)

Struktur:



Designmønsteret Proxy
(fortsat)

Deltagere:

Proxy (f.eks. `ServiceProvider_Stub`), der har en reference, der kan bruges til at referere det virkelige objekt, og som implementerer *Subject*-grænsefladen

Subject (f.eks. `Contract`), der definerer en fælles grænseflade for *Subject* og *Proxy*, således at *Proxy* kan bruges overalt, hvor *Subject* forventes

RealObject (f.eks. `ServiceProvider`), der definerer det virkelige objekt, som *Proxy* repræsenterer

Udvikling af et RMI-baseret chat-system

Fjernmetodekald på både server- og klientside.

Server:

- login
- logout
- sendMessage

Client:

- receiveLogin
- receiveLogout
- receiveMessage

Grænseflader

```
public interface ChatServerInterface extends Remote {
    public void login(String name, ChatClientInterface newClient)
                    throws RemoteException;
    public void logout(String name) throws RemoteException;
    public void sendMessage(Message message) throws RemoteException;
}
```

```
public interface ChatClientInterface extends Remote {
    public void receiveLogin(String name) throws RemoteException;
    public void receiveLogout(String name) throws RemoteException;
    public void receiveMessage(Message message) throws RemoteException;
}
```

Message

```
public class Message implements java.io.Serializable {
    public String name, text;

    public Message(String name, String text) {
        this.name = name;
        this.text = text;
    }
}
```

ChatServer

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

public class ChatServer extends UnicastRemoteObject
                        implements ChatServerInterface {
    Map<String, ChatClientInterface> chatters =
        new HashMap<String, ChatClientInterface>();

    public ChatServer() throws RemoteException {}

    public synchronized void login(String name,
           ChatClientInterface newClient) throws RemoteException {
        chatters.put(name, newClient);
        for (ChatClientInterface client : chatters.values())
            client.receiveLogin(name);
    }
```

fortsættes

```
    public synchronized void logout(String name)
                            throws RemoteException {
        chatters.remove(name);
        for (ChatClientInterface client : chatters.values())
            client.receiveLogout(name);
        System.out.println("client " + name + " logged out");
    }

    public synchronized void sendMessage(Message message)
                            throws RemoteException {
         for (ChatClientInterface client : chatters.values())
             client.receiveMessage(message);
    }

    public static void main(String[] args) {
        try {
            LocateRegistry.createRegistry(1099);
            Naming.rebind("ChatServer", new ChatServer());
        } catch (Exception e) { e.printStackTrace(); }
    }
}
```

ChatClient

```
public class ChatClient extends UnicastRemoteObject
                        implements ChatClientInterface {
    String name;
    ChatServerInterface server;
    ChatFrame gui;

    public ChatClient(String name, String url) throws RemoteException {
        this.name = name;
        try {
            server = (ChatServerInterface)
                java.rmi.Naming.lookup("rmi://" + url + "/ChatServer");
            server.login(name, this);
        } catch (Exception e) { e.printStackTrace(); }
        gui = new ChatFrame(this);
    }
```

fortsættes

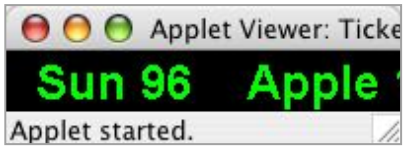
```
public void receiveLogin(String name) {
    gui.output.append(name + " entered\n");
}

public void receiveLogout(String name) {
    gui.output.append(name + " left\n");
}

public void receiveMessage(Message message) {
    gui.output.append(message.name + ": " + message.text + "\n");
}
```

fortsættes

```
    void sendTextToChat(String text) {
        try {
            server.sendMessage(new Message(name, text));
        } catch (RemoteException e) { e.printStackTrace(); }
    }

    void disconnect() {
        try {
            server.logout(name);
        } catch (Exception e) { e.printStackTrace(); }
    }

    public static void main(String[] args) {
        if (args.length != 2)
            throw new RuntimeException(
                "Usage: java ChatClient <user> <host>");
        try {
            new ChatClient(args[0], args[1]);
        } catch (RemoteException e) { e.printStackTrace(); }

    }
}
```

Børsnoteringer

Studér lærebogens design af client/server-programmel til løbende underretning om aktiekurser.



Client pull:

Klienterne kontakter serveren med passende mellemrum for at få oplyst de aktuelle kurser

Client push:

Serveren underretter interesserede klienter, når kurserne ændres

Socket-baserede udgaver: s. 600 - 614
RMI-baserede udgaver: s. 620 - 628

JDBC

Java **D**ata**b**ase **C**onnectivity

JDBC er en API, der tillader et Java-program at tilgå relationelle databaser.

JDBC tilbyder en simpel grænseflade til SQL (**S**tructured **Q**uery **L**anguage).

Muliggør:

- skabelse af en database
- indsættelse, søgning og opdatering
- tilgang til metadata

Eksempel på en database

name	address	city	state	zip	email	creditcard	orderno
M. Jordan	549 E. Surf	Chicago	IL	60657	mjordan@nba.com	Visa	102219985502
S. Pippen	11 S. 59th	Oakbrook	IL	60606	spippen@nba.com	Discover	103119983212
D. Rodman	234 N. 3rd	Chicago	IL	60030	drodman@nba.com	MasterCard	103119982223
S. Kerr	4010 Pine	Los Angeles	CA	90507	skerr@nba.com	Visa	100219991201
R. Harper	234 N. 3rd	New York	NY	20304	rharper@nba.com	MasterCard	012019992013

Basale SQL-kommandoer

`INSERT`	Indsætter en eller flere nye rækker i en tabel
`DELETE`	Sletter en eller flere rækker i en tabel
`UPDATE`	Modificerer værdierne i en eller flere rækker i en tabel
`SELECT`	Udvælger information baseret på en betingelse
`CREATE`	Skaber et nyt databaseobjekt
`DROP`	Fjerner et eksisterende databaseobjekt
`ALTER`	Ændrer formatet for et eksisterende databaseobjekt

JDBC-drivere

En driver udgør bindeledet imellem program og database.

Der er fire typer:

(1) JDBC-ODBC-bro: -> ODBC (Open Data Base Connectivity)
(2) Native-API (delvis Java): -> DBMS
(3) Net-protocol (ren Java): -> netprotokol -> DBMS
(4) Native-protocol (ren Java): -> DBMS' netprotokol

Ikke-kommercielle JDBC-drivere

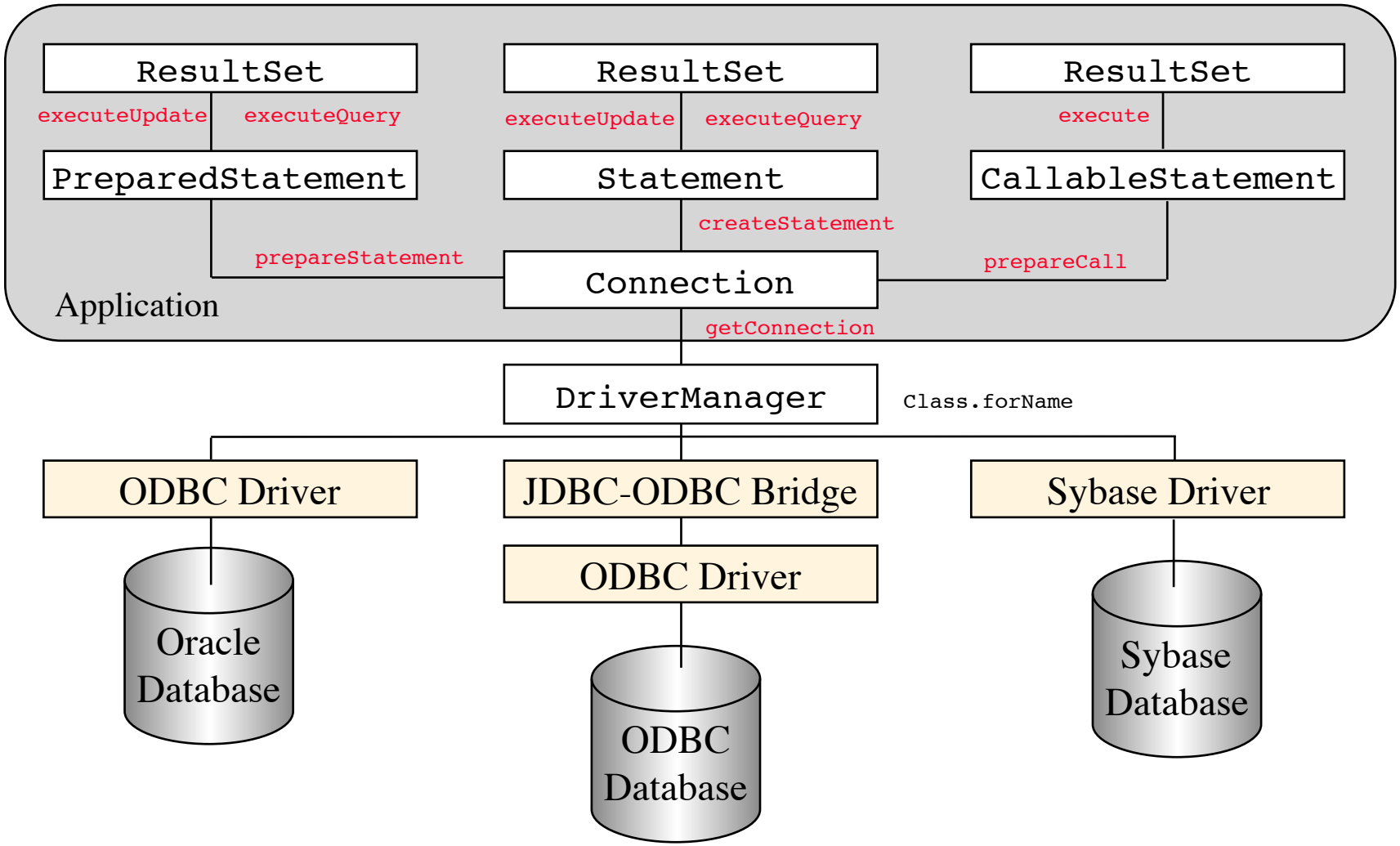
(open source type 4 drivere)

MySQL  http://www.mysql.com/

Cloudscape  http://www.cloudscape.com/

PostgreSQL  http://www.postgresql.org/

JDBC-interaktion



Skabelse af en JDBC-tabel

1. Driveren hentes

```
import java.sql.*;
import java.io.*;

public class BuildBeetlesTable {
    public static void main(String args[])
                                throws SQLException, IOException {
        System.out.println("Loading JDBC driver...");
        try {
            Class.forName("com.mysql.jdbc.Driver");
        } catch(ClassNotFoundException e) {
            System.exit(1);
        }
```

fortsættes

2. Forbindelse til database etableres

```
System.out.println("Connecting to Orders database...");
String url = "jdbc:mysql://localhost/OrdersDriver";
String user = "keld";
String password = "xyz123";
Connection conn = DriverManager.getConnection(url,
                                              user,
                                              password);
conn.setAutoCommit(false);
```

fortsættes

3. En tabel skabes

```
System.out.print("Building new Beetles table...\n\n");
Statement stmt = conn.createStatement();
String createString =
    "CREATE TABLE Beetles (name VARCHAR(35), " +
                          "address VARCHAR(35),  "  +
                          "city VARCHAR(20), "  +
                          "state VARCHAR(2), "  +
                          "zip VARCHAR(5), "  +
                          "email VARCHAR(30), "  +
                          "creditCard VARCHAR(15))";
stmt.executeUpdate(createString);
```

fortsættes

4. En række indsættes

```
System.out.print("Inserting test row in Beetles table...\n\n");
String insertString =
    "INSERT INTO Beetles VALUES ('Michael Owen', " +
                                "'123  Elmwood  Street',  "  +
                                "'Lake  Forest', "  +
                                "'IL', "  +
                                "'65431', "  +
                                "'mowen@aol.com', "  +
                                "'MasterCard')";
stmt.executeUpdate(insertString);
```

fortsættes

5. Tabellens indhold udskrives

```
ResultSet rset = stmt.executeQuery("SELECT * FROM Beetles");
while (rset.next()) {
    System.out.print("     ");
    System.out.print(rset.getString("name") + ", ");
    System.out.print(rset.getString(2) + ", "  );
    System.out.print(rset.getString(3) + ", "  );
    System.out.print(rset.getString(4) + ",\n      ");
    System.out.print(rset.getString(5) + ", ");
    System.out.print(rset.getString(6) + ", ");
    System.out.print(rset.getString(7) + "\n");
}
```

fortsættes

6. Ressourcer frigives, og forbindelsen til databasen lukkes

```
        System.out.println("Closing database connection...");
        conn.commit();    // make changes permanent
        stmt.close();
        rset.close();
        conn.close();     // release DBMS ressources
    }
}
```

Modifikation af en JDBC-tabel

1. Driveren hentes

```
import java.sql.*;
import java.io.*;
import java.util.Calendar;

public class ModifyBeetlesTable {
    public static void main(String args[])
                            throws SQLException, IOException {
        System.out.println("Loading JDBC driver...");
        try {
            Class.forName("com.mysql.jdbc.Driver");
        } catch(ClassNotFoundException e) {
            System.exit(1);
        }
```

fortsættes

2. Forbindelse til database etableres

```
System.out.println("Connecting to Beetles table...");
String url = "jdbc:mysql://localhost/OrdersDriver" +
             "?user=keld&password=xyz123";

Connection conn = DriverManager.getConnection(url);
conn.setAutoCommit(false);
```

fortsættes

3. Der tilføjes et ekstra felt (søjle)

```
stmt = conn.createStatement();
System.out.print("Modifying Beetles table...\n\n");
String updateString =
    "ALTER TABLE Beetles ADD ORDERNO VARCHAR(12)”;
stmt.executeUpdate(updateString);
```

fortsættes

4. En række i tabellen opdateres

```
Calendar cal = Calendar.getInstance();
String orderNo = String.valueOf(cal.get(cal.MONTH) + 1) +
                 String.valueOf(cal.get(cal.DAY_OF_MONTH)) +
                 String.valueOf(cal.get(cal.YEAR)) +
                 String.valueOf(cal.get(cal.HOUR)) +
                 String.valueOf(cal.get(cal.MINUTE)) +
                 String.valueOf(cal.get(cal.SECOND));
updateString = "UPDATE Beetles SET ORDERNO = " + orderNo +
               " WHERE NAME='Michael Owen'";
stmt.executeUpdate(updateString);
```

fortsættes

5. Tabellens indhold udskrives

```
System.out.print("Displaying table contents...\n\n");
ResultSet rset = stmt.executeQuery("SELECT * FROM Beetles");
while (rset.next()) {
     System.out.println("     " +
                        rset.getString("name") + ", " +
                        rset.getString("address") + ", " +
                        rset.getString("city") + ", " +
                        rset.getString("state") + "\n     " +
                        rset.getString("zip") + ", " +
                        rset.getString("email") + ", " +
                        rset.getString("creditCard") + ", " +
                        rset.getString("orderno") );
}
```

fortsættes

6. En række slettes

```
System.out.println("Deleting test record...");
updateString = "DELETE FROM Beetles WHERE NAME='Michael Owen'";
stmt.executeUpdate(updateString);
```

fortsættes

7. Ressourcer frigives, og forbindelsen til databasen lukkes

```
        System.out.println("Closing database connection...");
        conn.commit();
        stmt.close();
        rset.close();
        conn.close();
    }
}
```

E-handel med en ordre-database


```
import java.util.Calendar;
import java.awt.event.*;
import javax.swing.*;
import java.sql.*;
import java.io.*;

public class JdbcOrderDialog extends OrderDialog {
    public JdbcOrderDialog(JFrame owner) {
        super(owner);
    }

    protected ActionListener makeButtonHandler() {
        return new JdbcButtonHandler();
    }

    class JdbcButtonHandler implements ActionListener { ... }
}
```

fortsættes

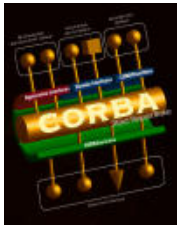
```
class JdbcButtonHandler implements ActionListener {
    public void actionPerformed(ActionEvent evt) {
        JButton button = (JButton) evt.getSource();
        String label = button.getText();
        if (label.equals("Ok") {
            try {
               insertNewOrder();
            } catch (IOException IOex) {}
            dialogPanel.reset();
            setVisible(false);
        }
    }

    public void insertNewOrder()
        throws SQLException, IOException { ... }
}
```

fortsættes


```
public void insertNewOrder() throws SQLException, IOException {
    System.out.print("Loading JDBC driver...\n\n\n" );
    try {
        Class.forName("com.mysql.jdbc.Driver");
    } catch (ClassNotFoundException e) {
        System.exit(0);
    }
    String url = "jdbc:mysql://localhost/OrdersDriver" +
                 "?user=keld&password=xyz123";
    Connection conn = DriverManager.getConnection(url);
    Statement stmt = conn.createStatement();
    String creditCard = null;
    if (dialogPanel.visaBox.isSelected())
        creditCard = "Visa";
    else if (dialogPanel.mcBox.isSelected())
        creditCard = "MasterCard";
    else if (dialogPanel.discoverBox.isSelected())
        creditCard = "Discover";
    // ...
```

fortsættes

```
        Calendar cal = Calendar.getInstance();
        String orderNo = String.valueOf(cal.get(cal.MONTH)+1) +
                         String.valueOf(cal.get(cal.DAY_OF_MONTH)) +
                         String.valueOf(cal.get(cal.YEAR)) +
                         String.valueOf(cal.get(cal.HOUR)) +
                         String.valueOf(cal.get(cal.MINUTE)) +
                         String.valueOf(cal.get(cal.SECOND));
        String insertString =
            "INSERT INTO Beetles VALUES( " +
            "'" + dialogPanel.nameField.getText()    + "'," +
            "'" + dialogPanel.addressField.getText() + "'," +
            "'" + dialogPanel.cityField.getText()    + "'," +
            "'" + dialogPanel.stateField.getText()   + "'," +
            "'" + dialogPanel.zipField.getText()     + "'," +
            "'" + dialogPanel.emailField.getText()   + "'," +
            "'" + creditCard                         + "'," +
            "'" + orderNo                            + "')";
        stmt.executeUpdate(insertString);
        stmt.close();
        conn.commit();
        conn.close();
    }
```

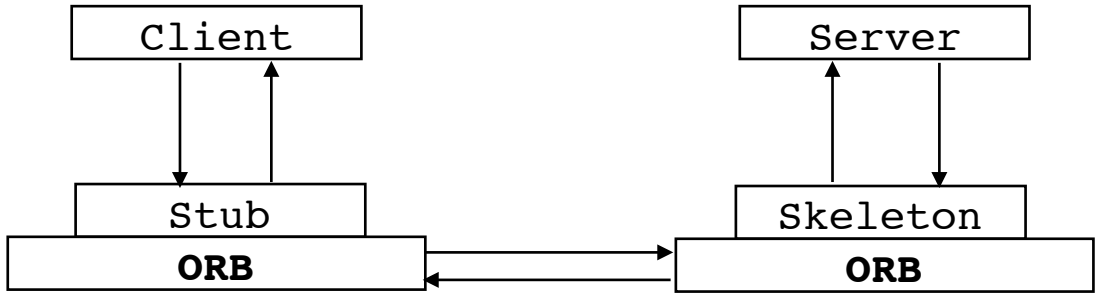


CORBA

Common **O**bject **R**equest **B**roker **A**rchitecture

CORBA er et framework, der muliggør samspil imellem objekter, der er implementeret i forskellige sprog.

Arkitekturen er helt analog til RMI's.



Serverkontrakten defineres i IDL (**I**nterface **D**efinition **L**anguage)

Andre Java-teknologier

JavaBeans

Et framework til at definere genbrugelige program-komponenter, der kan manipuleres visuelt. Teknologien baseres på overholdelsen af en række navnekonventioner.

Enterprise JavaBeans

Udvidelse af JavaBeans med beholder-baserede objekter, der kan kommunikere indbyrdes via netværk.

Java Media Framework

- Java Sound
- Java Speech
- Java Telephony
- Java 2D
- Java 3D
- Java Advanced Imaging

Java Cryptography Extension

Understøtter fortrolighed ved kommunikation over netværk

Jini

Gør det muligt for forskellige digitale enheder let og hurtigt at udveksle tjenester i et netværk.

Ugeseddel 9

26. oktober - 2. november

- Intet læsestof og ingen øvelsesopgaver.
- Der arbejdes med afleveringsopgaven.
- Næste kursusgang afholdes opgaveseminar og kursusevaluering. Hver opgavegruppe bedes forberede en 10 minutters fremlæggelse:
    1. Hvad går opgaven ud på?
    2. Hvor langt er gruppen nået?
    3. Hvilke delopgaver mangler at blive løst?