CS 580 Client-Server Programming
Spring Semester, 2007

Doc 14 Streamless Connections, DB & Architecture
March 22, 2007

References

Patterns of Enterprise Application Architecture, Martin Folwer, Addison-Wesley, 2003

Streamless Socket Access

Reading/writing on sockets without streams

Provides access to more socket functionality

NIO & Sockets

Important new classes

- Channels
- Buffers
- Encoders
- Decoders

New packages

- java.nio
- java.nio.channels
- java.nio.charset

Channels

Two-way connection to an IO device

Has

- Blocking IO
- Multiplexed non-blocking IO with selectors

Supports

- Sockets
- Files
- Pipes

Buffers

Channels read/write into buffers

Buffer class for each primitive data type

Byte, int, float, char, double, long, short

Encoders & Decoders

Maps Unicode strings to/from bytes

Date Server Example

```
import java.io.*;
import java.net.*;
import java.nio.*;
import java.nio.channels.*;
import java.nio.charset.*;
import java.util.*;
public class NIOTimeServer {
    private ServerSocketChannel acceptor;
    private static Charset usAscii = Charset.forName("US-ASCII");
    private static CharsetDecoder asciiDecoder = usAscii.newDecoder();
    private static CharsetEncoder asciiEncoder = usAscii.newEncoder();
```

```
public static void main(String[] args) throws IOException {
   int port = Integer.parseInt( args[0]);

   NIOTimeServer server = new NIOTimeServer( port );
   server.run();
}

public NIOTimeServer(int port ) throws IOException {
   InetSocketAddress serverAddress =
      new InetSocketAddress(InetAddress.getLocalHost(), port);
   acceptor = ServerSocketChannel.open();
   acceptor.socket().bind( serverAddress );
}
```

Date Server Example

```
public void run()  {
      while (true)  {
         try {
            SocketChannel client = acceptor.accept();
            processRequest( client );
          }
         catch (IOException acceptError){
            // for a later lecture
          }
        }
  }

void processRequest( SocketChannel  client) throws IOException {
   try  {
      String request = readLine( client );
      String response = processRequest( request);
      CharBuffer charsOut = CharBuffer.wrap( response + "\r\n");
      ByteBuffer bytesOut = asciiEncoder.encode(charsOut);
      client.write(bytesOut);
   }
   finally  {  client.close(); }
  }
```

Date Server Example

```
String readLine( SocketChannel  client) throws IOException {
   ByteBuffer inputBytes = ByteBuffer.allocate(1024);
   String input = "";
   CharBuffer inputChars;
   while (input.lastIndexOf( "\n") < 0 ) {
      inputBytes.clear();
      client.read( inputBytes );
      inputBytes.flip();
      inputChars = asciiDecoder.decode(inputBytes);
      input = input + inputChars.toString();
    }
    return input;
 }

 String processRequest( String request ) {
    if (request.startsWith("date"))
       return new Date().toString();
    else
       return "";
   }
}
```

Databases and Architecture

Databases & Architecture

How to keep SQL isolated?

How to isolate database connection details?

How to keep dealing with the database under control?

How to structure programs that use databases?

Example – Office Hours

Common Operations

Find Office hours for instructor X
Find office hours of any graduate advisor
Find office hours of any undergraduate advisor
Find office hours of any TA
Who has office hours at time X
What times are there no office hours
Add office hours
Modify office hours

Tables

Faculty			
Id	Name	Office	Phone
1	Eckberg	GMCS-543	594-6834
2	Donald	GMCS-541	594-7248
3	Carroll	GMCS-537	594-7242

RoleTypes	
ID	Role
1	Undergraduate Advisor
2	Graduate Advisor
3	TA

OfficeHours				
Id	StartTime	EndTime	Day	FacultyId
1	10:00	11:00	Tuesday	1
2	10:00	11:00	Thursday	1

Roles	
FacultyId	TypeId
1	2
2	2
3	1

Issues about Database Connections

Database usernames and passwords should not be scattered in code

How much database connection detail should be scattered in the code

DatabaseConnector

```
public class DatabaseConnector {
        private String databaseUrl;
        private String user;
        private String password;
        private ArrayList connectionPool;

        private static DatabaseConnector instance =
                DatabaseConnector(“filename”);

        public static DatabaseConnector instance() {
                return instance;
        }

        private DatabaseConnector(String filename) {
                read file for database info
                set private fields
        }

        public ResultSet executeQuery( String sql ) {
                return getStatement().executeQuery( sql);
        }

        public Statement getStatement() {
                return getConnection().createStatement();
        }

        private Connection getConnection() {
                return a connection
        }
        etc
}
```

Organizing Domain Logic

How to organize an application that uses a database

Fowler provides the following methods

Transaction Script
Domain Model
Table Module
Service Layer

Transaction Script

Identify different transactions to be performed by the application

Each transaction is handled by a separate method

Consequences

Very simple to implement

As application grows in complexity, becomes overly complex and hard to manage

Transaction Script

```
public class VoteData {
        public boolean addName(String name, etc) {
                code & SQl to add name to database }

        public boolean voteFor(String name) {
                code & SQL to vote for a poll
        }

        public addPoll(poll data) {
                add a poll to the database
        }
```

Domain Model

Implement classes that incorporates both behavior & data

Classes represent objects in the domain

Program becomes collection of interacting objects

Objects map to tables

A single object may span many tables

A table row may contain multiple objects

Consequences

Overly complex for simple applications

Scales well to complex applications

Database organizes data differently

Table Module

For each table (or view) implement a class

Each class holds the business logic related to the data in the table

Consequences

Classes are organized around database structure rather than OO principles

Handles more complex situations than Transaction Script

Not as scalable as Domain Model

Organizing Access to Database

Table Data Gateway
Row Data Gateway
Active Record
Data Mapper

Table Data Gateway

One object handles all the rows in a table or view

Each table has one class that knows the table

One object represents the table – all the rows

Gateway hides all the Sql from the rest of the program

Works well with

- Table Module
- Transaction Script

OfficeHours Gateway

```
public class OfficeHoursGateway {

        private static     String addOfficeHoursSql =
                        “INSERT
                          INTO officeHours ( startTime, endTime, day, facultyId )
                          VALUES ( ? , ?, ‘?’, ?)”;

        Private static String officeHoursSql =
                        “SELECT startTime, endTime, day
                        FROM officeHours
                        WHERE facultyId = ?”;

      public ResultSet officeHoursFor(int facultyId,) {
            Statement hoursStatement = DatabaseConnector.instance().
                              prepareStatement(officeHoursSql);
            hoursStatement.setObject( 1, facultyId);
            return hoursStatement.executeQuery();
      }
```

OfficeHours Gateway

```
public int setOfficeHoursFor(int facultyId, Time start, Time end, String day) {

        Statement addOfficeHours = DatabaseConnector.instance().
            prepareStatement(addOfficeHoursSql);
        addOfficeHours.setObject(1, start);
        addOfficeHours.setObject(2, end);
        addOfficeHours.setObject(3, day);
        addOfficeHours.setObject(4, facutlyId);
        return addOfficeHours.executeQuery();
}
```

Transaction Script + Table Gateway

```
public class OfficeHoursServer {
        private OfficeHoursGateway officeHours;
        private FacultyGateway faculty;
        etc.

        public Vector officeHoursFor(String facultyName) {
                int facultyId = faculty.idFor(facultyName,);

                ResultSet officeHoursRows = officeHours.officeHoursFor( facultyId);
                Vector officeHours = new Vector();
                while (officeHoursRows.next() ) {
                        Dictionary officeHour = new Dictionary();
                        officeHour.put( “start”, officeHoursRows.getObject( “start”));
                        officeHour.put( “end”, officeHoursRows.getObject( “end”));
                        officeHour.put( “day”, officeHoursRows.getObject( “day”));
                        officeHours.add( officeHour);
                }
                officeHoursRows.close();
                return officeHours;
        }
        etc.
}
```

Row Data Gateway

One object handles or represents a single row in a table or view

Each table has one class that knows the table

Gateway hides all the Sql from the rest of the program

A class provides just accessor methods to data in a row

Works well with Transaction script

sdsu.sql.DatabaseTable

Utility for Row Access

Part of SDSU Java library

Some Creation methods

```
Connection db;
db = DriverManager.getConnection( dbUrl, user, password);

DatabaseTable rows;

//Get rows from table Faculty with column Name = Donald
rows = DatabaseTable.getRow("Faculty", "Name", "Donald", db);
rows.elementAt(rowIndex, "Office");

// Get rows returned from a SQL select statment
rows = DatabaseTable.fromSQL("a SQL select", db);
```

Active Record

Each domain object know how add/remove/find its state in the database

In simple cases

- Class for each table
- An object represents one row in the table
- Similar to Row Data Gateway with domain logic

Faculty

```
public class Facutly {
        String name;
        String phoneNumber;
        int id;
        etc.

        private final static String findByNameSql =
                "SELECT *
                 FROM faculty
                WHERE name = '?'";

        public static Faculty findByName(String name ) {
                Statement find =
                        dabaseConnnector.prepareStatement(findByNameSql);
                find.setObject( 1, name);
                ResultSet facultyRow = find.executeQuery();
                return load(facultyRow);
}

public static Faculty load( ResultSet facultyRow) {
        create faculty object.
        get data out of Resultset.
        Put data into faculty object.
        Return faculty object.
}
```

Faculty

```
public boolean hasOfficeHoursAt(Time anHour) {
        Iterator hours = officeHours().iterator();
        while (hours.hasNext() ) {
                OfficeHour officeHour = (OfficeHour) hours.next();
                if (officeHour.contains( anHour) ) return true;
        }
        return false;
}

public ArrayList officeHours() {
        if( officeHours = nil ) {
                officeHours = OfficeHour.findFor( id );
        }
        return officeHours;
}
```

Domain Model + Active Record

```
public class OfficeHoursServer {

    public ArrayList officeHoursFor(String facultyName) {

        Faculty X = Faculty.findByName (facultyName,);

        ArrayList officeHours = X.officeHours();

            Convert contents of officeHOurs to XML-RPC acceptable types
            return vector of valid XML-RPC types;
    }

    etc.
}
```