Videregående programmering i Java

Dag 11 - Persistens

Fremlæggelse af programmering/status for projekter - medbring klassediagram og skærmbillede

Projektvejledning

JNI og kald til maskinkode/C/C++ fra Java

Designmønstrenes brug i standardbibliotekerne

Persistens og strategier til databaseadgang, evt.: http://hibernate.org/

Evt.: Objektpersistens og JDO - Java Data Objects

Internationalisering

Læsning: VP 7, VP 8, VP 10, evt. PRO kap.9 (om JNI)

Sidste gang - spørgsmål?

JUnit – obligatorisk i jeres projekt
Demo: Test af GUI
Hvordan lave Test af GUI om til tutorial

Persistensproblemet
Strategier til databaseadgang

Rigtig mange muligheder, f.eks.:

- JDBC-kode blandet sammen med resten af koden
- JDBC-kode i dedikerede klasser
- JDBC RowSet - ResultSet, der også opdaterer databasen
  - CachedRowSet - kan eksistere løsrevet fra databasen
  - WebRowSet giver XML, der kan transformeres, f.eks. med
    - XSLT (XSL style sheet)
    - Javas XML-behandling
- EJB - Enterprise JavaBeans - ét serverobjekt pr. række
- JDO - Java Data Objects - ét objekt pr. række
- Hibernate - http://hibernate.org/
- Proprietære løsninger
  - Giver mulighed for grafiske databasekomponenter
  - Oracle JDeveloper ADF - Application Developer Framework
  - Borland JBuilder: DataModule
  - IBM WebSphere Studio

Bedste løsning? Afhænger af omstændighederne!

- Hvor omfattende databasedelen af ens applikation er
- Hvor meget man forventer den senere skal vedligeholdes.

JDBC – databaseadgang

Indlæse driveren

Med Java under Windows følger en standard JDBC-ODBC-bro med, så man kan kontakte alle datakilder, der er defineret under ODBC:

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

Er det en anden database, skal man have en jar-fil med en driver fra producenten. Nyeste drivere kan findes på http://java.sun.com/jdbc/

Driver til en Oracle-database (hedder typisk classes12.zip):

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

Driver til en MySQL-database (hentes på http://mysql.com):

```
Class.forName("com.mysql.jdbc.Driver")
```

Etablere forbindelsen

Herefter kan man oprette forbindelsen med (for en ODBC-driver):

```
Connection forb = DriverManager.getConnection("jdbc:odbc:datakilde1");
```

Datakildens navn (her "datakilde1") skal være defineret i Windows.

Oracle-database:

```
Connection forb = DriverManager.getConnection(
  "jdbc:oracle:thin:@ora.javabog.dk:1521:student","jacob","jacob");
```

MySQL-database:

```
DriverManager.getConnection("jdbc:mysql:///jacob","root","xyz");
```

Databasedrivere

JDBC-drivere findes i fire typer:

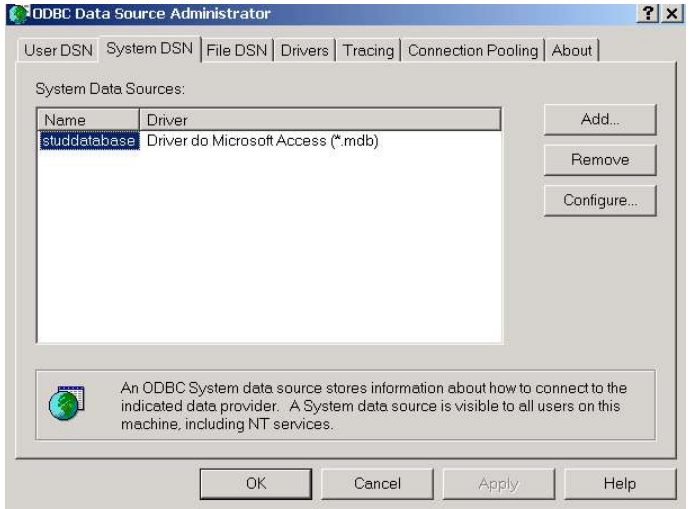
- Type 1: JDBC-ODBC-broen. Langsomste og kun til Windows.
- Type 2: Drivere skrevet i C eller C++ til den specifikke platform (normalt de hurtigste).
- Type 3: Platformsuafhængig (ren Java-) driver med databaseuafhængig kommunikationsprotokol
- Type 4: Platformsuafhængig (ren Java-) driver skrevet til at kommunikere med en specifik database (mest udbredte og næsten lige så hurtig som type 2).

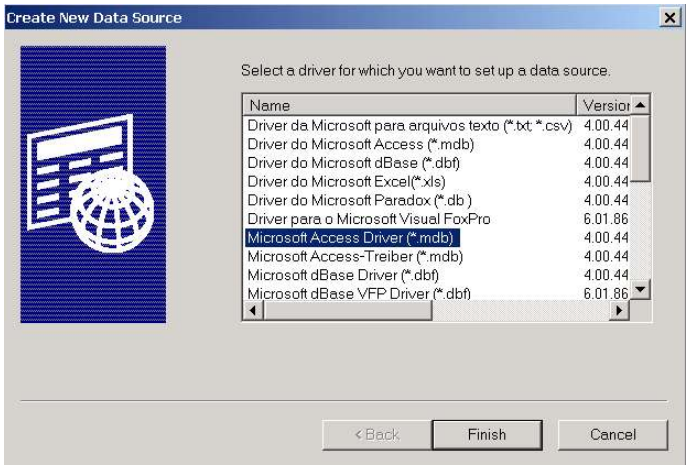
JDBC – databaseadgang

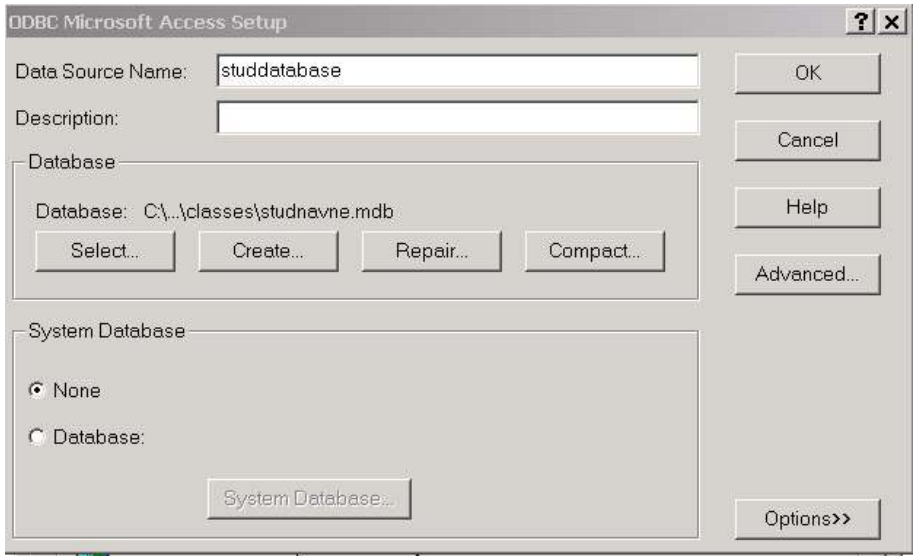
Lave JDBC-ODBC-bro til Access-fil

Eksempel:

1. Denne computer
2. Kontrolpanel
3. Administration
4. 

5. 

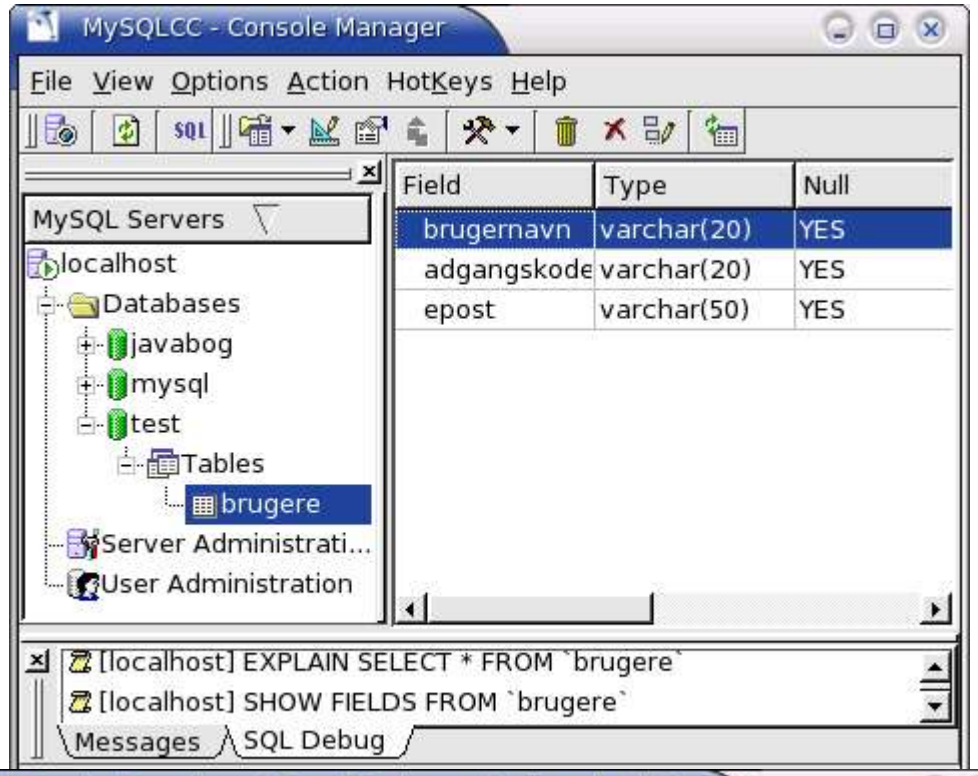
6. 

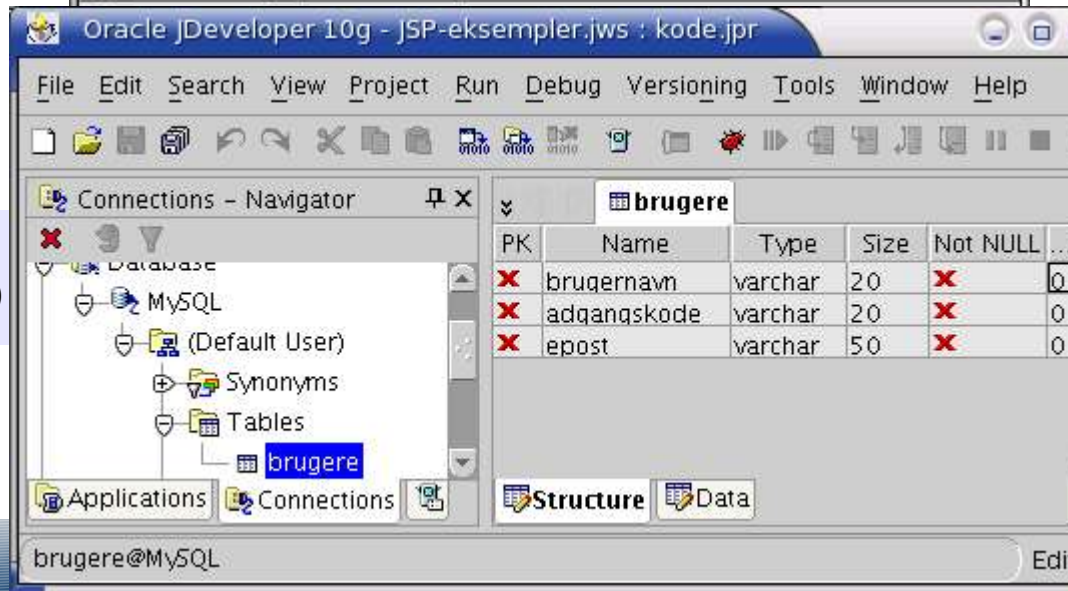
7. 

Forbindelse til database (MySQL)

- Installér MySQL
  - Hent fra mysql.com
  - test-database god i starten
  - Grafiske værktøjer
- Installér JDBC-driver
  - Connector/J fra mysql.com
  - Læg JAR-fil i java/jre/lib/ext/
- Kontakt test-database:

```
Class.forName("com.mysql.jdbc.Driver");
Connection forb =
  DriverManager.getConnection("jdbc:mysql:///test")
```





Forberedte SQL-kommandoer

```
import java.sql.*;
public class ForberedtSQL {
    public static void main(String[] arg) throws Exception
    {
       Class.forName("com.mysql.jdbc.Driver");
       Connection con = DriverManager.getConnection("jdbc:mysql:///test");

       // Forbered kommandoerne til databasen, f.eks. i starten af programmet:
       PreparedStatement indsætPstm = con.prepareStatement(
                                  "INSERT INTO kunder (navn,kredit) VALUES(?, ?)");

       PreparedStatement hentPstm = con.prepareStatement(
                                  "SELECT navn, kredit FROM kunder WHERE navn=?");

       // under programudførelsen kan de forberedte kommandoer udføres mange gange:
       for (int i=0; i<100; i++)
       {
          indsætPstm.setString(1, "Brian");
          indsætPstm.setInt(2, i);
          indsætPstm.execute();

          indsætPstm.setString(1, "Hans' venner");  // bemærk ' i strengen
          indsætPstm.setInt(2, 1042+i);
          indsætPstm.execute();


          hentPstm.setString(1, "Hans' venner");   // bemærk ' i SQL-forespørgslen
          ResultSet rs = hentPstm.executeQuery();

          // man løber igennem svaret som man plejer
          while (rs.next())
          {
             String navn = rs.getString(1);
             double kredit = rs.getDouble(2);
             System.out.println(navn+" "+kredit);
          }
       }
    }
}
```

Samlede batch-opdateringer

```
import java.sql.*;
public class Batchopdateringer
{
    public static void main(String[] arg) throws Exception
    {
      Class.forName("com.mysql.jdbc.Driver");
      Connection con = DriverManager.getConnection("jdbc:mysql:///test");

      PreparedStatement pstmt = con.prepareStatement(
                          "INSERT INTO kunder (navn,kredit) VALUES(?, ?)");

      pstmt.setString(1, "Hans");
      pstmt.setInt(2, 142);
      pstmt.addBatch();

      pstmt.setString(1, "Grethe");
      pstmt.setInt(2, 242);
      pstmt.addBatch();

      // send ændringer til databasen
      pstmt.executeBatch();
    }
}
```

Uden automatisk commit

```
import java.sql.*;
public class UdenAutocommit
{
    public static void main(String[] arg) throws Exception
    {
      Class.forName("oracle.jdbc.driver.OracleDriver");
      Connection forb = DriverManager.getConnection(
                "jdbc:oracle:thin:@ora.javabog.dk:1521:student","jacob","jacob");

      try {
        forb.setAutoCommit(false);
        Statement stmt = forb.createStatement();

        stmt.executeUpdate("insert into KUNDER(NAVN,KREDIT) values('Jacob',-17)");
        stmt.executeUpdate("insert into KUNDER(NAVN,KREDIT) values('Brian', 0)");

        // flere transaktioner ...
        System.err.println("Alt gik godt, gør ændringerne forpligtigende");
        forb.commit();
      }
      catch (Exception e)
      {
        e.printStackTrace();
        System.err.println("Noget gik galt! Annullerer ændringerne...");
        forb.rollback();
      }
      finally
      {
        // Husk at sætte auto-commit tilbage, af hensyn til andre transaktioner
        forb.setAutoCommit(true);
      }
    }
}
```

Metadata

```
import java.sql.*;
public class BenytMetadataOgUdskrivTabel
{
    /** Udskriver et ResultSet pænt. Finder selv ud af kolonnenavnene. */
    public static void udskriv(String titel, ResultSet rs) throws Exception {
       ResultSetMetaData rsmd = rs.getMetaData();
       int antalKolonner = rsmd.getColumnCount();
       System.out.println();
       System.out.println(" --------- " + titel + " ("+antalKolonner+" kolonner)");

       // udskriv kolonnenavnene
       for (int i=1; i<=antalKolonner; i++) skrivFormateret(rsmd.getColumnName(i));
       System.out.println();

       // udskriv cellerne i hver række
       while (rs.next())
       {
          for (int i=1; i<=antalKolonner; i++) skrivFormateret(""+rs.getString(i));
          System.out.println();
       }
       System.out.println(" ----------------------------");
    }

    public static void main(String[] arg) throws Exception
    {
       Class.forName("oracle.jdbc.driver.OracleDriver");
       Connection forb = DriverManager.getConnection(
                 "jdbc:oracle:thin:@ora.javabog.dk:1521:student","jacob","jacob");

       DatabaseMetaData dmd = forb.getMetaData();
       System.out.println("DatabaseProductName = "+dmd.getDatabaseProductName());
       System.out.println("DriverName          = "+dmd.getDriverName());
       System.out.println("MaxRowSize          = "+dmd.getMaxRowSize());

       ResultSet rs = dmd.getTables(null, "JANO", "%", null);
       udskriv("tabeller i databasen", rs);

       Statement stmt = forb.createStatement();
       rs = stmt.executeQuery("select * from KUNDER");
       udskriv("kunder", rs);
    }

    public static void skrivFormateret(String str) {
```

Metadata

```
import java.sql.*;
public class BenytMetadataOgUdskrivTabel
{
    /** Udskriver et ResultSet pænt. Finder selv ud af kolonnenavnene. */
    public static void udskriv(String titel, ResultSet rs) throws Exception {
       ResultSetMetaData rsmd = rs.getMetaData();
       int antalKolonner = rsmd.getColumnCount();
       System.out.println();
       System.out.println(" --------- " + titel + " ("+antalKolonner+" kolonner)");

       // udskriv kolonnenavnene
       for (int i=1; i<=antalKolonner; i++) skrivFormateret(rsmd.getColumnName(i));
       System.out.println();

       // udskriv cellerne i hver række
       while (rs.next())
       {
          for (int i=1; i<=antalKolonner; i++) skrivFormateret(""+rs.getString(i));
          System.out.println();
       }
       System.out.println(" ------
    }

    public static void main(String
    {
       Class.forName("oracle.jdbc.
       Connection forb = DriverMan
                 "jdbc:oracle:thi

       DatabaseMetaData dmd = forb
       System.out.println("Databas
       System.out.println("DriverN
       System.out.println("MaxRowS

       ResultSet rs = dmd.getTables(null, "JANO", "%", null);
       udskriv("tabeller i databasen", rs);

       Statement stmt = forb.createStatement();
       rs = stmt.executeQuery("select * from KUNDER");
       udskriv("kunder", rs);
    }

    public static void skrivFormateret(String str) {
```

```
DatabaseProductName = Oracle
DriverName          = Oracle JDBC driver
MaxRowSize          = 2000

 --------- tabeller i databasen (5 kolonner)
TABLE_CAT     | TABLE_SCHEM   | TABLE_NAME    | TABLE_TYPE    | TABLE_REMARKS |
null          | JANO          | KUNDER        | TABLE         | null          |
null          | JANO          | PERSONER      | TABLE         | null          |
 ------------------------------

 --------- kunder (2 kolonner)
NAVN          | KREDIT        |
Jacob         | -1799         |
Brian         | 0             |
Hans          | 500           |
 ------------------------------
```

Opdatere og navigere i ResultSet

De fleste ResultSet har metoder til at bevæge sig aktivt rundt i svaret og endda opdatere databasen gennem svaret.
Det gøres med f.eks.:

```
Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                                     ResultSet.CONCUR_UPDATABLE);
ResultSet rs = stmt.executeQuery("select NAVN, KREDIT from KUNDER");
```

Derefter kan man navigere rundt i ResultSet-objektet med f.eks.:

```
rs.absolute(3);     // går til 3. række i svaret (regnet fra 1 af)
rs.previous();      // går en række tilbage (modsatte af next())
rs.first();         // går til starten af svaret (svarende til rs.absolute(1))
rs.relative(3);     // går 3 rækker frem, dvs. til 4. række
int r = rs.getRow();// giver hvilken række vi nu er i (her returneres 4)
```

Man kan ændre i data med f.eks.:

```
rs.updateString("NAVN", "Jakob"); // ændrer kundens navn til Jakob
rs.updateRow();                   // opdaterer rækken i databasen

rs.moveToInsertRow();             // flyt til speciel indsættelses-række
rs.updateString("NAVN", "Søren"); // sæt navn
rs.updateDouble("KREDIT", 1000);  // sæt kredit
rs.insertRow();                   // indsæt rækken i databasen
rs.moveToCurrentRow();            // gå væk fra speciel indsættelsesrække
```

Derudover findes cancelRowUpdates(), der annullerer opdateringer i en række, deleteRow(), der sletter den aktuelle række fra både svaret og databasen, refreshRow(), der opfrisker ResultSet-objektet med de nyeste data.

JDBC RowSet

- RowSet = et sæt rækker hentet fra databasen
  - =ResultSet+oprindelse+ændringer
- RowSet-objekter giver mulighed ny arbejdsform
  - Man behøver kun at spørge på data (f.eks. med SELECT) hvorefter man får et RowSet-objekt.
    - Når man skal opdatere nogle rækker, slette eller oprette nye, opererer man blot det eksisterende RowSet-objekt
    - Slut med at rode med INSERT-, DELETE- eller UPDATE-sætninger i SQL.
- Flere undertyper af RowSet i JDK 1.5
  - JdbcRowSet =ResultSet+oprindelse+ændringer
  - CachedRowSet =rækkerne er gemt i hukommelsen
    - kan altså eksistere afbrudt fra databasen!
      - Undertyper: FilteredRowSet og JoinRowSet
  - WebRowSet =CachedRowSet repræsenteret som XML

JdbcRowSet

```
import java.sql.*;
import javax.sql.*;
import javax.sql.rowset.*;
import com.sun.rowset.*; // Bemærk: rowset.jar fra Sun skal være i CLASSPATH

public class BenytJdbcRowSet
{
   public static void main(String[] arg) throws Exception
   {
      Class.forName("com.mysql.jdbc.Driver");
      Connection con = DriverManager.getConnection("jdbc:mysql:///test");
      JdbcRowSetImpl jrs = new JdbcRowSetImpl(con);

      // Lav forespørglen
      jrs.setCommand("SELECT * FROM kunder WHERE navn = ?");
      jrs.setString(1,"Jacob");
      jrs.execute();

      // Udskriv resultatet
      while (jrs.next())
      {
         String navn = jrs.getString("navn");
         double kredit = jrs.getDouble("kredit");
         System.out.println(navn+" "+kredit);
      }
   }
}
```

CachedRowSet

```
import java.sql.*;
import javax.sql.*;
import javax.sql.rowset.*;
import com.sun.rowset.*; // Bemærk: rowset.jar fra Sun skal være i CLASSPATH

public class BenytCachedRowSet
{
  public static void main(String[] arg) throws Exception
  {
    Class.forName("com.mysql.jdbc.Driver");
    Connection con = DriverManager.getConnection("jdbc:mysql:///test");
    Statement stmt = con.createStatement();
    ResultSet rs = stmt.executeQuery("SELECT * FROM kunder");

    CachedRowSetImpl crs = new CachedRowSetImpl();
    crs.populate(rs);
    rs.close();

    // Opdatér første række i CachedRowSet-objektet
    crs.first();
    crs.updateDouble("kredit", -2000);
    crs.updateRow();

    // Indsæt række
    crs.moveToInsertRow();
    crs.updateString("navn", "Poul Nyrup");
    crs.updateDouble("kredit", 100000);
    crs.insertRow();
    crs.moveToCurrentRow();

    // Opdatér data i databasen
    crs.setUrl("jdbc:mysql:///test");
    crs.setUsername("root");
    crs.setPassword("");
    crs.acceptChanges();
  }
}
```

WebRowSet

```
public class BenytWebRowSet
{
  public static void main(String[] arg) throws Exception
  {
    Class.forName("com.mysql.jdbc.Driver");
    Connection con = DriverManager.getConnection("jdbc:mysql:
    Statement stmt = con.createStatement();
    ResultSet rs = stmt.executeQuery("SELECT * FROM kunder");

    WebRowSetImpl wrs = new WebRowSetImpl();
    wrs.populate(rs);
    rs.close();

    // Generér XML
    wrs.writeXml(System.out);

    // Opdatér første række i WebRowSet-objektet
    wrs.first();
    wrs.updateDouble("kredit", -3000);
    wrs.updateRow();

    // Indsæt række
    wrs.moveToInsertRow();
    wrs.updateString("navn", "Fogh");
    wrs.updateDouble("kredit", 72);
    wrs.insertRow();
    wrs.moveToCurrentRow();

    // Generér XML der nu også omfatter ændringerne
    wrs.writeXml(System.out);

    // Opdatér data i databasen
    wrs.setUrl("jdbc:mysql:///test");
    wrs.setUsername("root");
    wrs.setPassword("");
    wrs.acceptChanges();
  }
}
```

```
<?xml version="1.0"?>
<webRowSet xmlns="http://java.sun.com/xm
  <metadata>
    <column-count>2</column-count>
    <column-definition>
      <column-index>1</column-index>
      <column-display-size>32</column-di
      <column-name>navn</column-name>
      <table-name>kunder</table-name>
      <column-type-name>VARCHAR</column-
    </column-definition>
    <column-definition>
      <column-index>2</column-index>
      <column-name>kredit</column-name>
      <schema-name></schema-name>
      <table-name>kunder</table-name>
      <column-type-name>FLOAT</column-ty
    </column-definition>
  </metadata>
  <data>
    <currentRow>
      <columnValue>jacob</columnValue>
      <columnValue>-2000.0</columnValue>
    </currentRow>
    <currentRow>
      <columnValue>brian</columnValue>
      <columnValue>0.0</columnValue>
    </currentRow>
    <currentRow>
      <columnValue>Poul Nyrup</columnVal
      <columnValue>100000.0</columnValue
    </currentRow>
  </data>
</webRowSet>
```

Evt.: JDO - Java Data Objects

```
import javax.jdo.*;

 private PersistenceManagerFactory pmf = null;
 // initialisering af pmf (udbyderspecifik, ikke vist)

  PersistenceManager pm;
  Transaction transaction;


  // oprette et objekt (beskrevet i XML-dokumentet) og gemme det
  Person p = new Person("Jacob", "Nordfalk", "nordfalk@mobilixnet.dk");
  Object id;

  pm = pmf.getPersistenceManager();
  transaction = pm.currentTransaction();
  pm.makePersistent(p);
  id = pm.getObjectId(p);
  transaction.commit();
  pm.close();

...

  // hente et objekt (der ikke skal ændres)
  pm = pmf.getPersistenceManager();
  p = (Person) pm.getObjectById(id, false);
  pm.close();
```

```
  // ændre objekter
  pm = pmf.getPersistenceManager();
  transaction = pm.currentTransaction();
  p = (Person) pm.getObjectById(id, false);
  person.setName("Troels");
  transaction.commit();
  pm.close();

...

  // slette objekter
  pm = pmf.getPersistenceManager();
  transaction = pm.currentTransaction();
  p = (Person) pm.getObjectById(id, false);
  pm.deletePersistent(p);
  transaction.commit();
  pm.close();
```

Hibernate

```java
import java.util.Date;

public class Event {
    private Long id;

    private String title;
    private Date date;

    Event() {}

    public Long getId() {
        return id;
    }

    private void setId(Long id) {
        this.id = id;
    }

    public Date getDate() {
        return date;
    }

    public void setDate(Date date) {
        this.date = date;
    }

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }
}
```

```xml
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-mapping
<hibernate-mapping>

    <class name="Event" table="EVENTS">
        <id name="id" column="EVENT_ID">
            <generator class="increment"/>
        </id>
        <property name="date" type="timestamp" column="EVEN
        <property name="title"/>
    </class>

</hibernate-mapping>
```

```java
Session session = HibernateUtil.currentSession();
Transaction tx = session.beginTransaction();

Event theEvent = new Event();
theEvent.setTitle("Besøge mor");
theEvent.setDate(new Date());

session.save(theEvent);

tx.commit();
HibernateUtil.closeSession();
```

Resumé af skabende designmønstre

- Fabrikeringsmetode
  - en metode, der fabrikerer objekter for klienten (i stedet for at klienten selv opretter dem med new)
- Fabrik
  - et objekt med en fabrikeringsmetode
- Singleton
  - sikring af, at der kun eksisterer ét objekt af en bestemt slags
- Abstrakt Fabrik/Toolkit
  - er en Fabrik med nedarvinger, der sørger for objektoprettelsen. Hvilken nedarving der anvendes, bestemmes af en fabrikeringsmetode
- Bygmester
  - simplificerer oprettelsen af nogle relaterede objekter ved at oprette og konfigurere objekterne (evt. trinvist) for klienten
- Prototype
  - objekter oprettes ud fra eksisterende skabelon-objekter
- Objektpulje
  - genbrug de samme objekter igen og igen ved at huske dem i en pulje

Hyppigt anvendte designmønstre

- Proxy
  - få metodekald til at gå gennem et mellem-objekt (proxyen), der modtager metodekald på vegne af (fungerer som en erstatning) for det rigtige objekt og kalder videre i det rigtige objekt
  - Herunder Virtuel Proxy/Doven Initialisering
    - udskyde oprettelsen af det rigtige objekt til første gang, der er brug for det
- Adapter
  - et hjælpeobjekt, der får et objekt til at passe ind i et system ved at fungere som omformer mellem objektet og systemet
- Iterator
  - hjælper med at gennemløbe nogle data
- Facade
  - forenkler brugen af et sæt objekter ved at give en simplificeret grænseflade til dem
- Observatør/Lytter
  - at objekter kan 'abonnere' på, at en ting (hændelse) sker
- Dynamisk Binding
  - at understøtte "plugin"-klasser, der kan indlæses under kørslen og dermed udvide programmet, efter at det er skrevet

Resumé af andre designmønstre

- Uforanderlig
  - objektet kan ikke ændres, når det først er oprettet
- Fluevægt
  - begræns antallet af objekter ved at sørge for, at der ikke bliver oprettet objekter med de samme data. I stedet er der mange referencer til de samme (unikke) objekter
- Filter
  - objekter, der "filtrerer" en strøm af data. Filtrene kan kombineres vilkårligt
- Lagdelt Initialisering
  - klienten opretter et objekt direkte, og dette objekt opretter eller fremskaffer det, der i virkeligheden skal bruges og videredelegerer det meste af arbejdet til det
- Komposit/Rekursiv Komposition
  - skabe et meget fleksibelt objekthierarki, ved at definere objekter, der kan indeholde andre objekter inkl. sin egen slags
- Kommando
  - registrér brugerens ændringer af data, sådan at de kan fortrydes igen

JDBC og dens brug af designmønstre

Fremlæggelser af projekt

Eksterne kald

Kald til eksterne programmer kan foregå på flere måder

- Starte ekstern proces
  - Process proces = Runtime.getRuntime().exec(program)
  - Nemt
  - Kommunikation gennem netværksport eller standard input og output
    - Begrænset kommunikation
    - Let fejlfinding
  - CPU-krævende hvis processen skal starte ofte
- JNI – Java Native Interface
  - Koden lænkes ind i programmet
  - Kommunikation sker ved almindelige metodekald
    - Objekt med metoder erklæret native
  - Sværere
    - Man skal kunne kode f.eks. C eller C++
    - Fejl kan få den virtuelle maskine til at gå ned
  - Meget effektivt
- Bindinger til specifikke systemer

Starte ekstern proces

```
import java.io.*;

public class KaldEksterntProgram
{
    public static void main(String[] args) throws Exception
    {
      // Start programmet 'sort', der sorterer linerne i en fil
      Process proces = Runtime.getRuntime().exec("sort");

      // sort kan læse data fra standard input og skriver dem på standard output
      PrintWriter s = new PrintWriter(proces.getOutputStream());
      BufferedReader l = new BufferedReader(
                         new InputStreamReader(proces.getInputStream()));
      s.println("En");
      s.println("snegl");
      s.println("på");
      s.println("vejen");
      s.println("er");
      s.println("tegn");
      s.println("på");
      s.println("regn");
      s.println("i");
      s.println("Spanien");
      s.close(); // luk datastrømmen (sort sorterer først når den har alle data)
      proces.waitFor(); // vent på at processen er færdig
      // Læs resultatet fra programmets standard output og udskriv det
      String lin;
      while ((lin = l.readLine()) != null) System.out.println("Fra sort: "+lin);
    }
}
```

```
Fra sort: En
Fra sort: er
Fra sort: i
Fra sort: på
Fra sort: på
Fra sort: regn
Fra sort: snegl
Fra sort: Spanien
Fra sort: tegn
Fra sort: vejen
```

Java-klasser med maskinkode

```
package vp;
public class HejVerdenFraCKode
{
    public native void hejVerden();
}
```

Generere vp_HejVerdenFraCKode.h
javah -jni vp.HejVerdenFraCKode

```
package vp;
public class BenytHejVerdenFraCKode
{
  public static void main(String[] args) throws Exception
  {
    System.out.println("Indlæser maskinkoden.");
    // Indlæs libHej.so (UNIX) eller hej.dll (Windows)
    System.loadLibrary("Hej");
    System.out.println("Opretter objekt.");
    HejVerdenFraCKode objekt = new HejVerdenFraCKode();
    System.out.println("Kalder metode i maskinkode.");
    objekt.hejVerden();
  }
}
```

```
#include "vp_HejVerdenFraCKode.h"
#include <stdio.h>

JNIEXPORT void JNICALL
Java_vp_HejVerdenFraCKode_hejVerden (
    JNIEnv *e, jobject o)
{
    printf("Hej Verden fra C++ !\n");
}
```

Oversæt C++-kode til .so-fil (eller .dll under Windows)
g++ -shared -g -I/usr/include/java vp_HejVerdenFraCKode.cpp -o libHej.so

Kør programmet
java -Djava.library.path=. vp.HejVerdenFraCKode

```
Indlæser maskinkoden.
Opretter objekt.
Kalder metode i maskinkode.
Hej Verden fra C++ !
```

Biblioteker til at gøre JNI lettere

- EasyJNI
- JACOB: JAVA-COM Bridge
  http://danadler.com/jacob/
- Jace
  http://reyelts.dyndns.org:8080/jace/release/docs/overview.html
  http://www.javaworld.com/javaworld/jw-05-2002/jw-0510-integrate.htm

Bindinger til specifikke systemer

- OpenOffice.org UNO
  - Styring af OpenOffice.org fra Java
    - Åbne dokumenter, indsætte/læse/formatere tekst etc
    - Demo
  - Brug af OpenOffice.org som komponent fra Java
    - Eksemplet er beregnet til videreudvikling
    - Demo

Udskrift til printer

- Simplest mulige udskrivning (fra JDK 1.1)

```
public void udskriv()
{
    java.awt.PrintJob pj = Toolkit.getDefaultToolkit().getPrintJob(this,"figurer",null);
    Graphics g = pj.getGraphics(); // Graphics-objekt til at tegne på papiret
    tegnFigurer(g);
    pj.end();  // Færdiggør siden med udskriften og send den til printeren
}
```

  - Kun én side
  - Meget lidt kontrol over sideformat
- Udskrift foregår ved tegning med et Graphics-objekt

Udskrift til printer

- Mere avanceret udskrivning (fra JDK 1.2)

```
public void udskriv()
{
    java.awt.print.PrinterJob pj = java.awt.print.PrinterJob.getPrinterJob();

    // implementér Printable-interfacet for at udskrive noget
    java.awt.print.Printable p = new java.awt.print.Printable() {
      public int print(Graphics g, java.awt.print.PageFormat pf, int side) {
        if (side > 4) return java.awt.print.Printable.NO_SUCH_PAGE;
        tegnFigurer(g);
        g.drawString("Side "+side, (int)pf.getImageableX(), (int)pf.getImageableY()+10);
        return java.awt.print.Printable.PAGE_EXISTS;
      }
    };                              // eller:
                                    //java.awt.print.Book b = new java.awt.print.Book();
                                    //b.append(p, pj.defaultPage());
    pj.setPrintable(p);             //pj.setPageable(b);

    pj.printDialog(); // vis printerdialog – hvor bruger kan vælge antal sider etc
    try {
      pj.print();    // udskriv!
    } catch (Exception ex) { ex.printStackTrace(); }
}
```

- Fuld kontrol over sideformat
- Flere sider (kan have forskellige sideformater)
- Udskrivningsdialog kender antal sider (med Pageable)

Udskrift til printer

- Java Print Service (fra JDK 1.3)
  - Søgning efter printere med bestemte evner
    - duplex, farve, PostScript, ...
  - Specificere formatet af det udskrevne
    - Opløsning (DPI), antal kopier, prioritet, ...
  - Sende data
    - Også i rå formater som
      - PostScript
      - PDF
      - GIF, JPEG, PNG

Eksempel: Udskrivning af PostScript-fil, 5 dobbeltsides A4-kopier:

```
FileInputStream psStream = new FileInputStream("file.ps");

DocFlavor psInFormat = DocFlavor.INPUT_STREAM.POSTSCRIPT;
Doc myDoc = new SimpleDoc(psStream, psInFormat, null);
PrintRequestAttributeSet aset = new HashPrintRequestAttributeSet();
aset.add(new Copies(5));
aset.add(MediaSize.A4);
aset.add(Sides.DUPLEX);
PrintService[] services = PrintServiceLookup.lookupPrintServices(psInFormat, aset);
if (services.length > 0) {
    DocPrintJob job = services[0].createPrintJob();
    job.print(myDoc, aset);
}
```

Mere læsning om udskrivning

Basal udskrivning

Udskrivning af flere sider
http://java.sun.com/developer/technicalArticles/Printing/Java2DPrinting/

Java Print Service
http://java.sun.com/j2se/1.4.2/docs/guide/jps/

Andre
http://java.sun.com/printing/

http://java.sun.com/developer/technicalArticles/Printing/

Internationalisering

- Når et program skal anvendes af flere kulturer og sprog, opstår behov for, at programtekster, beløb og dato angives i de pågældende landes sprog.

- *Internationalisering* (også kaldet I18N) består i at gøre programmet sprogneutralt, ved at sørge for at:
  - al formatering og fortolkning af tal-, beløbs-, dato- og tidsangivelser sker afhængigt af sproget
  - al sproglig tekst er flyttet til resursefiler.
  - Programmørs arbejde

- *Lokalisering* består i at:
  - oversætte resursefilerne til et bestemt sprog.
  - (ikke programmeringskyndig) oversætters arbejde

Internationalisering

```
import java.text.*;
import java.util.*;
public class BenytDateFormat
{
  public static void main(String arg[])
  {
    DateFormat klformat, datoformat, dkf;
    klformat   = DateFormat.getTimeInstance(DateFormat.MEDIUM);
    datoformat = DateFormat.getDateInstance(DateFormat.FULL);
    dkf = DateFormat.getDateTimeInstance(DateFormat.MEDIUM,DateFormat.SHORT);

    Date tid = new Date();
    System.out.println( tid );
    System.out.println( "Kl   :"+ klformat.format(tid) );
    System.out.println( "Dato :"+ datoformat.format(tid) );
    System.out.println( "Tid  :"+ dkf.format(tid) );
  }
}
```

Kørt på da nsk PC

```
Wed Feb 05 14:23:46 GMT+00:00 2003
Kl   :14:23:46
Dato :5. februar 2003
Tid  :05-02-2003 14:23
```

Kørt på amerikansk PC

```
Mon Dec 03 13:27:57 GMT+01:00 2001
Kl:   1:27:57 PM
Dato: Monday, December 3, 2001
Tid:  Dec 3, 2001 1:27 PM
```

Bruge Locale-objekter

```
import java.text.*;
import java.util.*;
public class BenytDateFormat2
{
  public static void main(String arg[]) {
    DateFormat klformat, datoformat;

    Locale fransk = new Locale("fr","FR");
    klformat   = DateFormat.getTimeInstance(DateFormat.SHORT,fransk);
    datoformat = DateFormat.getDateInstance(DateFormat.LONG, fransk);

    Date tid = new Date();
    System.out.println( "Kl:   "+ klformat.format(tid) );
    System.out.println( "Dato: "+ datoformat.format(tid) );
  }
}

Kl:   14:23
Dato: 5 février 2003
```

De mulige lokalindstillinger

```
import java.util.*;
public class MuligeSprog
{
  public static void main(String arg[]) {
    Locale[] lokalindstillinger = Locale.getAvailableLocales();
    for (int i=0; i<lokalindstillinger.length; i++) {
      Locale lokalindst = lokalindstillinger[i];
      System.out.print(lokalindst+" ");
      // udkommentér hvis du vil se hvad sproget hedder på sproget selv
      //System.out.println(lokalindst.getDisplayName(lokalindst));
    }
    System.out.println( );
  }
}

ar ar_AE ar_BH ar_DZ ar_EG ar_IQ ar_JO ar_KW ar_LB ar_LY ar_MA ar_OM ar_QA ar_SA
ar_SD ar_SY ar_TN ar_YE hi_IN iw iw_IL ja ja_JP ko ko_KR th th_TH th_TH_TH zh
zh_CN zh_HK zh_TW be be_BY bg bg_BG ca ca_ES cs cs_CZ da da_DK de de_AT de_CH
de_DE de_LU el el_GR en_AU en_CA en_GB en_IE en_IN en_NZ en_ZA es es_AR es_BO
es_CL es_CO es_CR es_DO es_EC es_ES es_GT es_HN es_MX es_NI es_PA es_PE es_PR
es_PY es_SV es_UY es_VE et et_EE fi fi_FI fr fr_BE fr_CA fr_CH fr_FR fr_LU hr
hr_HR hu hu_HU is is_IS it it_CH it_IT lt lt_LT lv lv_LV mk mk_MK nl nl_BE nl_NL
no no_NO no_NO_NY pl pl_PL pt pt_BR pt_PT ro ro_RO ru ru_RU sh sh_YU sk sk_SK sl
sl_SI sq sq_AL sr sr_YU sv sv_SE tr tr_TR uk uk_UA en en_US eo
```

- Locale-objektet består af tre dele:
  - Første del er sprogkoden, f.eks. da, sv, no, en, fr.
  - En valgfri anden del er landekoden, f.eks. DK, NO, GB, DE, FR
  - En valgfri tredje del er varianten inden for sprogområdet (f.eks. om valutaen er i euro).

Tekstindhold i resursefiler

```
package sprogtest;

import java.text.*;
import java.util.*;
public class BenytDateFormatMedResurser
{
  public static void main(String arg[])
  {
    ResourceBundle res = ResourceBundle.getBundle("sprogtest.Tekster");
    DateFormat klformat, datoformat, dkf;
    klformat   = DateFormat.getTimeInstance(DateFormat.MEDIUM);
    datoformat = DateFormat.getDateInstance(DateFormat.FULL);
    dkf = DateFormat.getDateTimeInstance(DateFormat.MEDIUM,DateFormat.SHORT);

    Date tid = new Date();
    System.out.println( res.getString("Kl_")+ klformat.format(tid) );
    System.out.println( res.getString("Dato_")+ datoformat.format(tid) );
    System.out.println( res.getString("Tid_")+ dkf.format(tid) );
  }
}
```

```
# sprogtest.Tekster.properties
Tid_=Tid  \:
Kl_=Kl   \:
Dato_=Dato \:
```

```
Kl  : 14:23:50
Dato: 5. februar 2003
Tid : 05-02-2003 14:23
```

```
# sprogtest.Tekster_en.properties
Tid_=Time       \:
Kl_=Time of day\:
Dato_=Date       \:
```

```
Time of day : 3:07:28 PM
Date        : Monday, December 3, 2001
Time        : Dec 3, 2001 3:07 PM
```