Java-opgraderingskursus

Danmarks Meteorologiske Institut

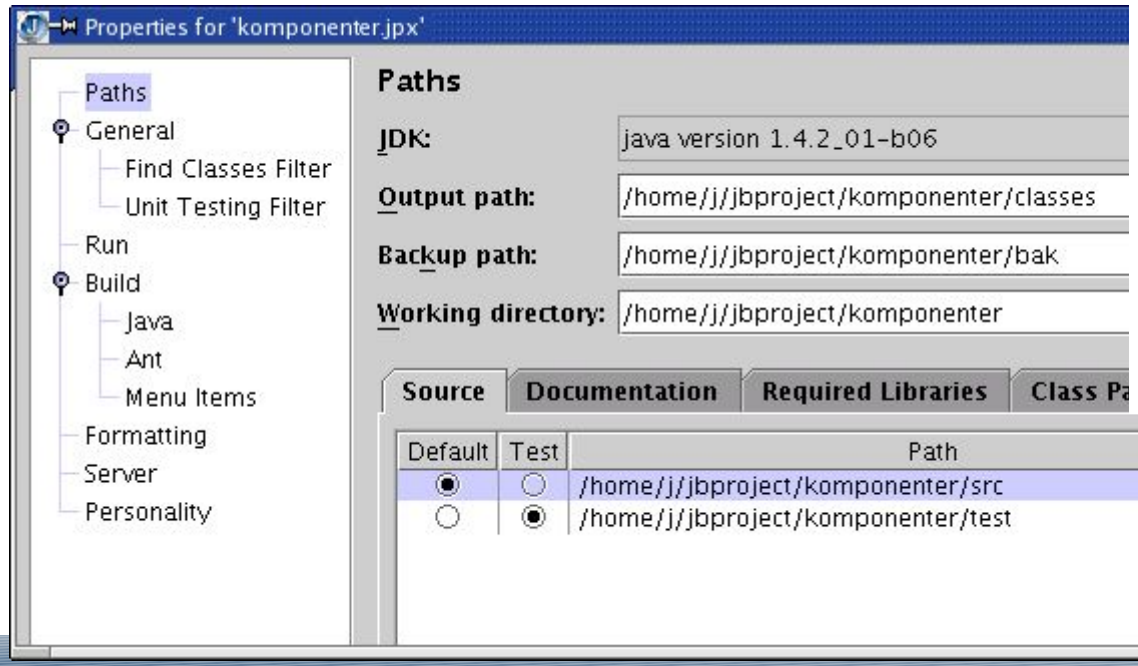
Et Javaprogram

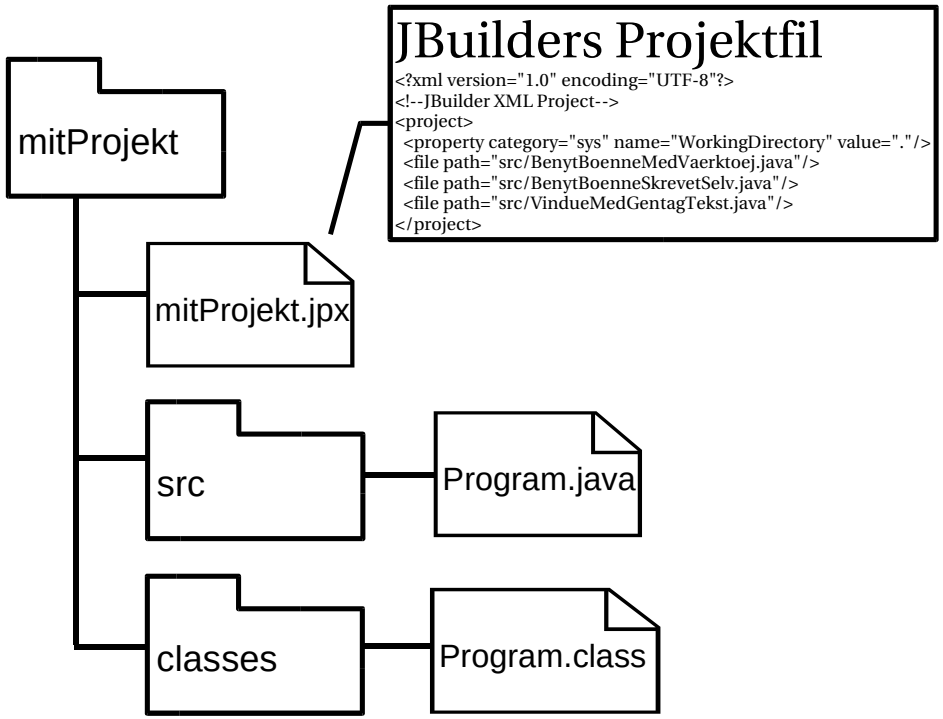
```
// Et simpelt program, der skriver "Hej verden"
// og et citat af Storm P. ud til skærmen
// filnavn: HejVerden.java
public class HejVerden
{
  public static void main (String[] arg)
  {
    System.out.println("Hej Verden!");
    System.out.println("Hvornår smager en Tuborg bedst?");
    System.out.println("Hvergang!");
  }
}
```

- Den binære kode - bytecode - i .class-filerne
  - Platformsuafhængig - dvs. ingen .exe-filer
  - Platformsafhængig optimering sker først under kørslen

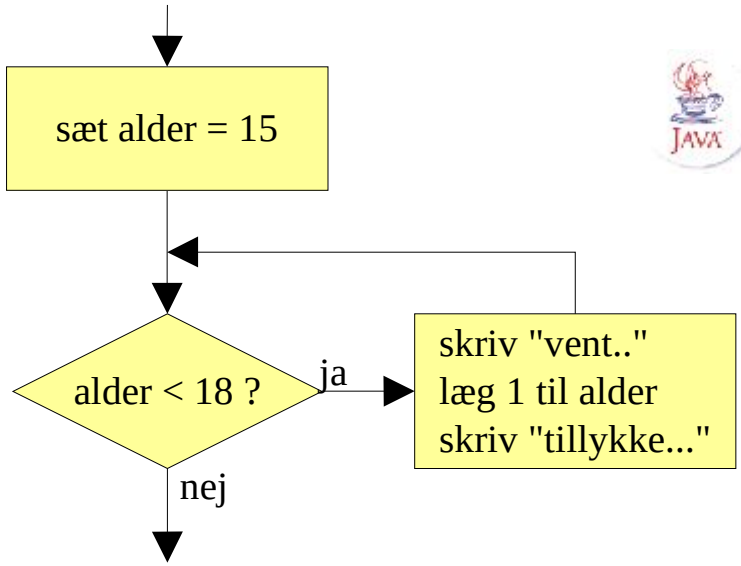
JBuilder og projekter

- Projektet holder rede på filer og egenskaber for programmet
  - ikke det samme som et katalog
  - projekter ligger ofte i C:\jbproject
- Problemer med stier?
  - Tjek at projektet forventer kildetekst det rigtige sted!
  - Tjek stierne i projektet
    - Åbn "Project Properties"
    - Kildekode i src/
    - .class-filer i classes/
  - Sti på .java-fil = src/ + pakke





Løkker




```
public class Alder4
{
  public static void main(String[] arg)
  {
    int alder;
    alder = 15;

    while (alder < 18)
    {
      System.out.println("Du er "+alder+" år. Vent til du bliver ældre.");
      alder = alder + 1;
      System.out.println("Tillykke med fødselsdagen!");
    }

    System.out.println("Nu er du "+alder+" år og myndig.");
  }
}
```

```
Du er 15 år. Vent til du bliver ældre.
Tillykke med fødselsdagen!
Du er 16 år. Vent til du bliver ældre.
Tillykke med fødselsdagen!
Du er 17 år. Vent til du bliver ældre.
Tillykke med fødselsdagen!
Nu er du 18 år og myndig.
```

De simple typer

Type	*Art*	*Antal bit*	*Mulige værdier*	*Standard-værdi*
byte	heltal	8	-128 til 127	0
short	heltal	16	-32768 til 32767	0
int	heltal	32	-2147483648 til 2147483647	0
long	heltal	64	-9223372036854775808 til 9223372036854775807	0
float	kommatal	32	±1.40239846E-45 til ±3.40282347E+38	0.0
double	kommatal	64	±4.94065645841246544E-324 til ±1.79769313486231570E+308	0.0
char	unicode	16	\u0000 til \uffff (0 til 65535)	\u0000
boolean	logisk	1	true og false	false

Samme størrelse på alle platforme (i modsætning til C og C++)

Hvor der muligvis mistes information, fordi intervallet af mulige værdier indsnævres, skal man skrive en eksplicit typekonvertering. F.eks:

```
int x;
double y;
y = 3.8;
x = (int) y;
```

Arrays

```
public class Maaneder
{
  public static void main(String[] arg)
  {
    int[] måneder = {31,28,31,30,31,30,31,31,30,31,30,31};

    System.out.println("Længden af januar er: " + måneder[0]);
    System.out.println("Længden af april er: " + måneder[3]);

    for (int i=0; i <måneder.length; i++)
      System.out.print(måneder[i] + ", ");

    System.out.println();
  }
}
```

```
Længden af januar er: 31
Længden af april er: 30
31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31,
```

main() har en parameter, som er et array af strenge. Dette array indeholder kommandolinie-argumenter ved kørsel af programmet.

```
public class Kommandolinie
{
  public static void main(String[] arg)
  {
    System.out.println("Antallet af argumenter er: " + arg.length);

    for (int i=0; i< arg.length; i=i+1)
      System.out.println("Argument "+i+" er: " + arg[i]);
  }
}
```

```
Antallet af argumenter er: 3
Argument 0 er: x
Argument 1 er: y
Argument 2 er: z
```

Programmet herover er kørt med "java Kommandolinie x y z".

Klassens anatomi

```
// Klassenavn skal være i filen Klassenavn.java

import klasser; // f.eks. import java.util.*;

public class Klassenavn
{
    // erklæring af variabler (og evt. samtidig initialisering)
    adgang  type   navnPåObjektvariabel;
    private String s2 = "goddag"; // samtidig initialisering

    // erklæring af konstruktører, evt. med parametre
    adgang Klassenavn(type1 parameter1, type2 parameter2, ...)
    {
       ... // kode der sætter objektvariablerne til startværdier
    }

    // erklæring af metoder, evt. med parametre
    adgang  returtype metodenavn(type1 parameter1, type2 parameter2, ...)
    {
       ... // kode
    }

    // eksempler på metoder:
    public int metode1()
    {
       return 15; // noget af type int
    }

    public void metode2(int nn, String ss)
    {
    }
}
```

```
public class Boks3
{
   private double længde, bredde, højde;

   public Boks3()
   {
      System.out.println("Standardboks oprettes");
      sætMål(10, 10, 10);
   }

   /** en anden konstruktør der får bredde, højde og længde */
   public Boks3(double lgd, double b, double h)
   {
      System.out.println("Boks oprettes med lgd="+lgd+" b="+b+" h="+h);
      sætMål(lgd,b,h);
   }

   public void sætMål(double lgd, double b, double h)
   {
      if (lgd<=0 || b<=0 || h<=0) {
         System.out.println("Ugyldige mål. Bruger standardmål.");
         længde = 10.0;
         bredde = 10.0;
         højde  = 10.0;
      } else {
         længde = lgd;
         bredde = b;
         højde  = h;
      }
   }

   public double volumen()
   {
      double vol = længde*bredde*højde;
      return vol;
   }
}
```

Boks3
-længde :double -bredde :double -højde :double
+Boks3() +Boks3(lgd, b, h) +sætMål(lgd, b, h) +volumen() :double


```
public class BenytBoks3
{
  public static void main(String[] arg)
  {
     Boks3 enBoks;
     enBoks = new Boks3();              // brug konstruktøren ude
     System.out.println("Volumen er: "+ enBoks.volumen());

     Boks3 enAndenBoks;
     enAndenBoks = new Boks3(5,5,10);  // brug den anden konstru
     System.out.println("Volumen er: "+ enAndenBoks.volumen());
  }
}
```

```
Standardboks oprettes
Volumen er: 1000.0
Boks oprettes med lgd=5.0 b=5.0 h=10.0
Volumen er: 250.0
```

Synlighed

public, protected og private

Variabler og metoder erklæret *public* er altid tilgængelige, både inden og uden for klassen.

Variabler og metoder erklæret *protected* er tilgængelige for alle klasser inden for samme pakke.
Klasser i andre pakker kan kun få adgang hvis de er nedarvinger.

Skriver man *ingenting* er det kun klasser i samme pakke der har adgang til variablen eller metoden.

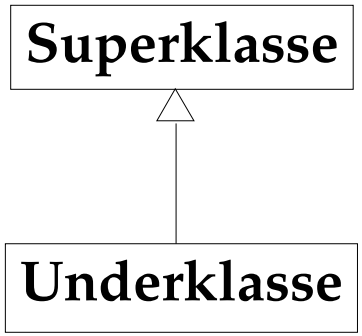
private er det mest restriktive. Hvis en variabel eller metode er erklæret private, kan den kun benyttes indenfor samme klasse (og derfor kan den ikke tilsidesættes med nedarving).

Adgang	***public***	***protected***	***(ingenting)***	***private***
i samme klasse	ja	ja	ja	ja
klasse i samme pakke	ja	ja	ja	nej
arving i en anden pakke	ja	ja	nej	nej
klasse der ikke arver i en anden pakke	ja	nej	nej	nej

Holder man sig inden for samme pakke er der altså ingen forskel mellem public, protected og ingenting.

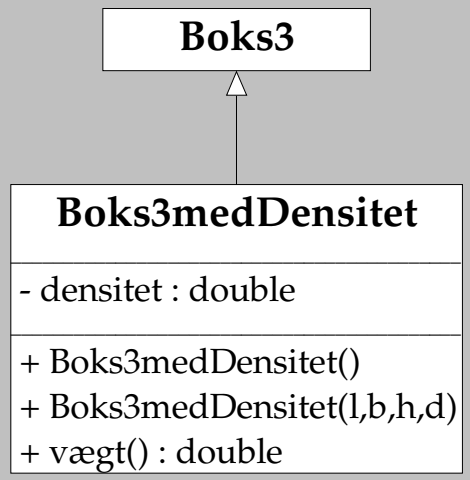
Arv

- En klasse kan arve variabler og metoder fra en anden klasse
- Klassen, der nedarves fra, kaldes superklassen
- Klassen, der arver fra superklassen, kaldes underklassen
- Underklassen kan tilsidesætte (omdefinere) metoder arvet fra superklassen ved at definere dem igen



Nedarving

```
public class Boks3medDensitet extends Boks3
{
    private double densitet;

    public Boks3medDensitet()
    {
       // super(); kaldes hvis intet andet angives
       densitet = 10.0;
    }

    public Boks3medDensitet(double lgd, double b,
              double h, double densitet)
    {
       // vælg en anden konstruktør i superklassen end den uden parametre
       super(lgd,b,h);
       this.densitet = densitet;
    }

    public double vægt()
    {
       return volumen() * densitet;    // superklassen udregner volumen for os
    }
}
```



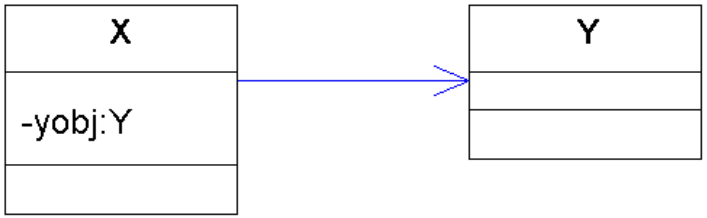
Arv og konstruktører

- Konstruktører skal defineres på ny i en underklasse
- En konstruktør kalder først en af superklassens konstruktører
- Superklassens konstruktør kan kaldes med: super(parametre)
- Hvis man ikke selv kalder en af superklassens konstruktører, indsætter Java automatisk et kald af superklassens konstruktør uden parametre

UML og klasserelationer

Har-relationen

At X *har* Y betyder: i klassen X er defineret en variabel af type Y. Et X-objekt har derfor en reference til et Y-objekt, og derfor har X-objektet mulighed for at kalde metoder og bruge variabler fra Y-objektet gennem denne reference. Y-objekter kender ikke nødvendigvis noget til X-objekters eksistens.

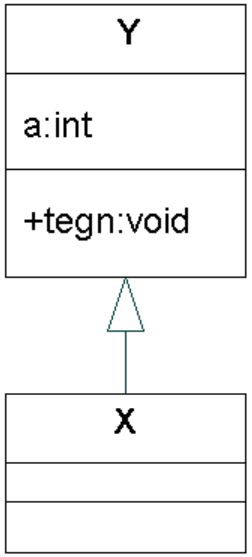


Eksempel:

```
public class X
{
   private Y yobj = new Y();
   ...
}
```

Er-en-relationen

At X *er-en* Y betyder: i klassen X er defineret, at X arver fra klassen Y. Et X-objekt indeholder derfor (mindst) alle de metoder og variabler, som et Y-objekt indeholder.

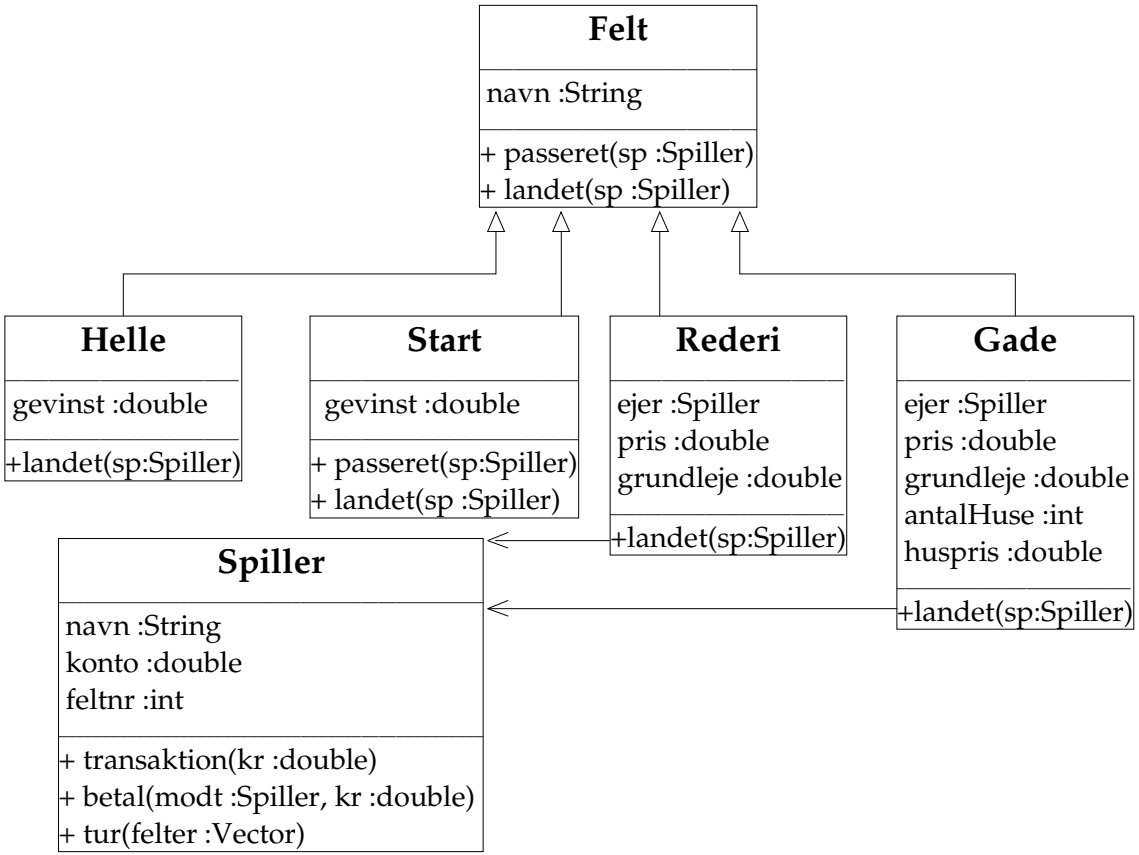


Eksempel:

```
public class X extends Y
{
   ...
}
```

Det er også en er-en-relation, når man implementerer et interface.

UML og klasserelationer

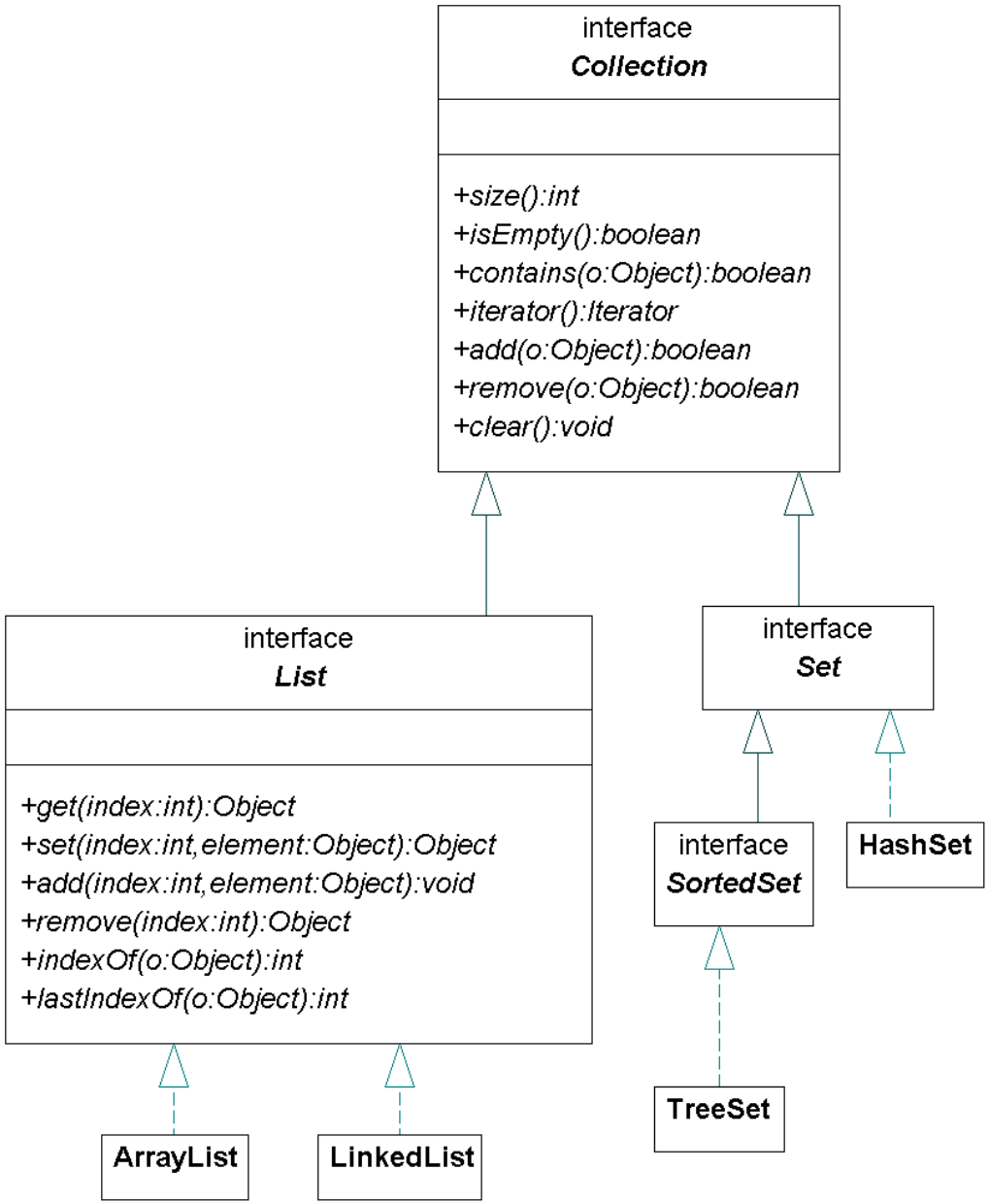


Rederi og Gade *har en* Spiller.

Helle, Start, Rederi og Gade *er-et* Felt.

Samlinger af data

- Collection: Samling af data
  - Indeholder metoder fælles for alle datastrukturer
- List: Ordnet liste
  - To klasser implementerer List: ArrayList og LinkedList.
- Set: Uordnet mængde
  - HashSet implementerer Set.
- SortedSet: Sorteret mængde
  - TreeSet implementerer Set.



Samlinger af data

Eksempel på brug af liste

```
List liste;
liste = new ArrayList();
liste.add("æh");      // alle samlinger af data
liste.add("øh",0);   // kun lister
```

Gennemløb v.hj.a. tællevariabel

```
for (int i=0; i<liste.size(); i++) {
          String s = (String) liste.get(i);
          // gør noget med s
          System.out.println(s);
}
```

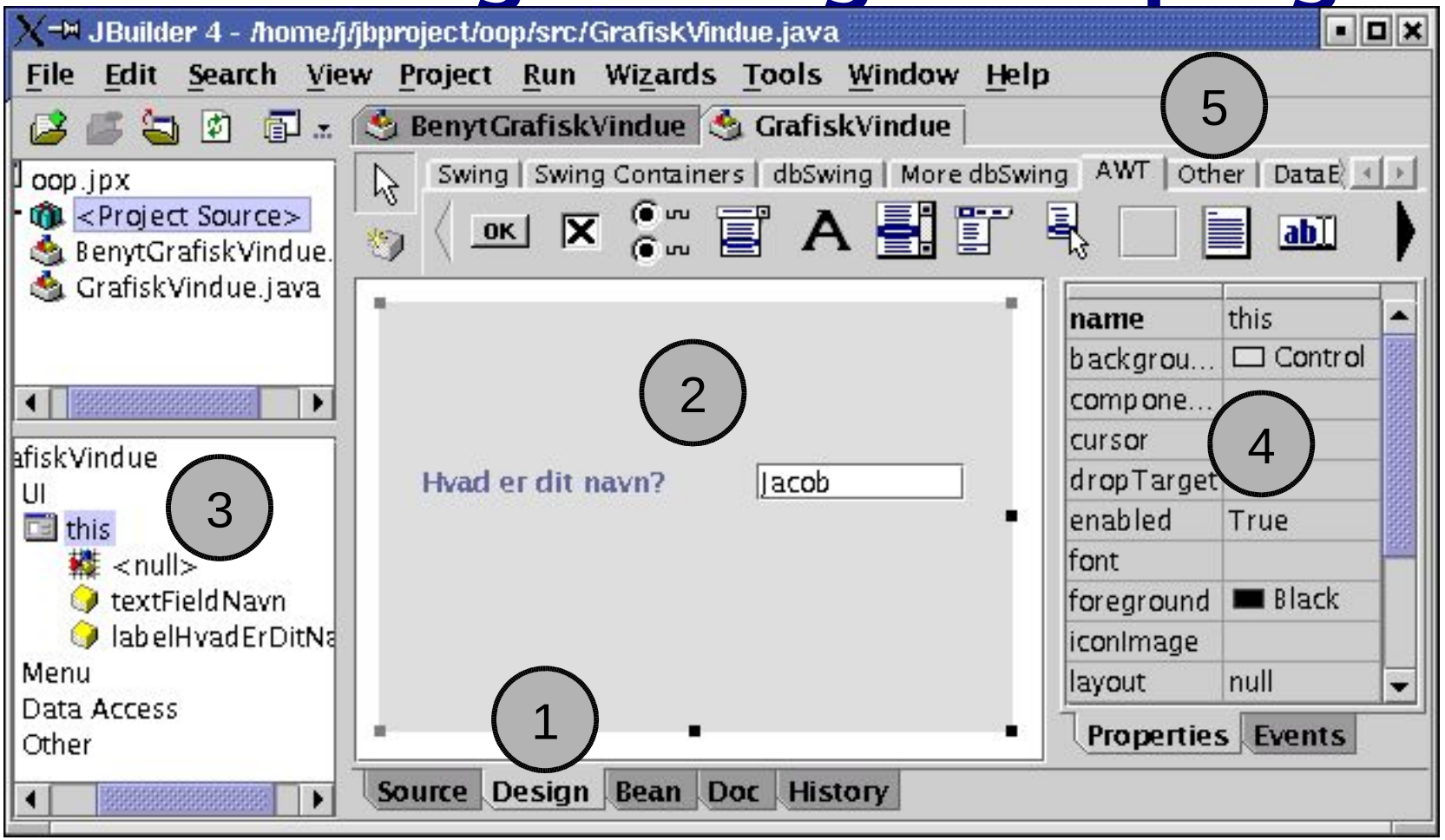
Et (lille) objekt, der hjælper med at gennemløbe data

Gennemløb v.hj.a. *iterator*

```
for (Iterator iter=liste.iterator(); iter.hasNext(); ) {
          String s = (String) iter.next();
          // gør noget med s
          System.out.println(s);
}
```

Man kan få et iterator-objekt ved at kalde iterator()
på enhver datastruktur (Collection)

Visuelt design af et grafisk program



(1) Design-fanen
(2) Visuelt designområde
(3) Struktur af programmet
(4) Egenskaber (og hændelser) på valgt element
(5) Komponentfaner

Genbrugelige komponenter

- Komponenter er programmørens byggeklodser
  - De kan bruges igen og igen i mange sammenhænge
  - De kan sættes sammen på alle mulige måder
  - De er nemme at indstille
  - Udviklingsværktøj kan generere programkoden for programmøren
- Grafiske brugergrænseflader er som regel helt bygget op af komponenter!
- Komponenter i Java hedder Javabønner (eng.: JavaBeans)

Komponentbaseret udvikling

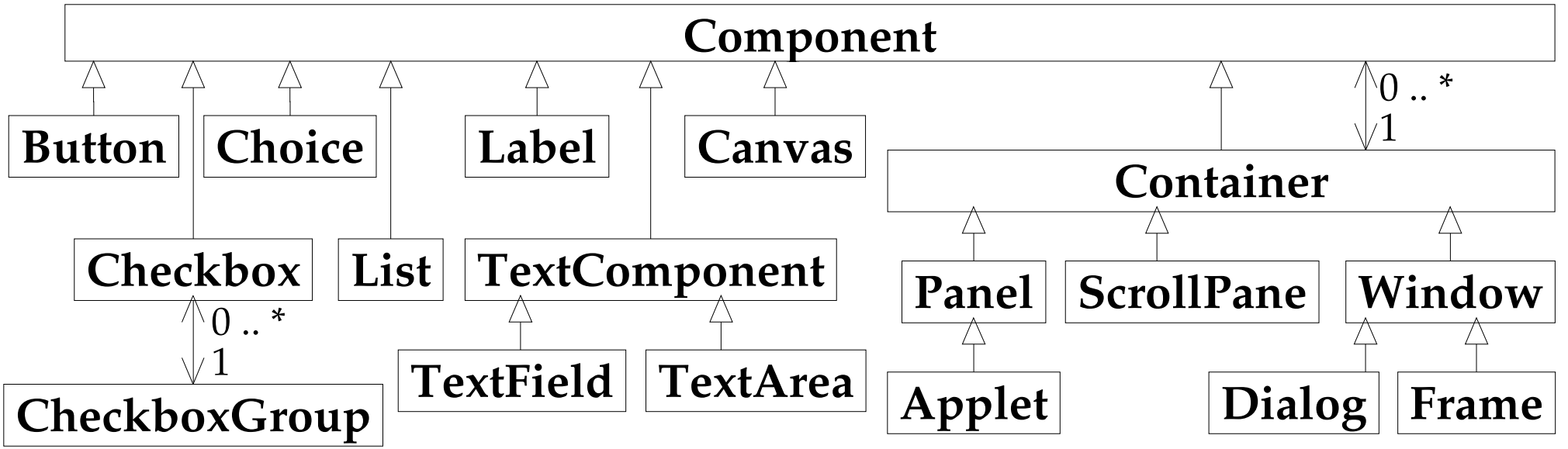
Eksempel: Bønnen TextField

Egenskab	*Type*	*Egenskab sættes med*	*Egenskab aflæses med*
text	String	setText(String t)	getText()
editable	boolean	setEditable(boolean rediger)	isEditable()
columns	int	setColumns(int bredde)	getColumns()
echoChar	char	setEchoChar(char tegn)	getEchoChar()

Bruge en javabønne fra et udviklingsværktøj

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

public class BenytBoenneMedVaerktoej extends Frame
{
   TextField textFieldNavn = new TextField();      // opret bønnen

   public BenytBoenneMedVaerktoej() {
      try {
         jbInit();
      }
      catch(Exception e) {
         e.printStackTrace();
      }
   }

   private void jbInit() throws Exception {
      textFieldNavn.setText("Jacob");               // sæt egenskaben text
      textFieldNavn.setBounds(new Rectangle(141, 61, 112, 29));
      this.setLayout(null);
      this.add(textFieldNavn, null);
   }
}
```

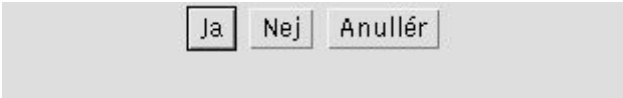
Komponenter og containere



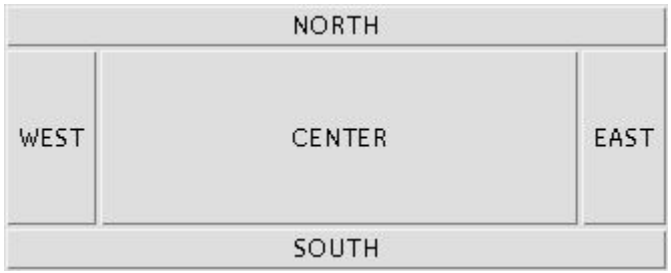
- Hule pile: *er-en*-relationer (dvs. nedarvning)
- De andre pile: *har*-relatioer
  - En container har nul til mange komponenter
  - Hver komponent har/tilhører én container
  - Tilsvarende med CheckboxGroup og Checkbox

Layout-managere

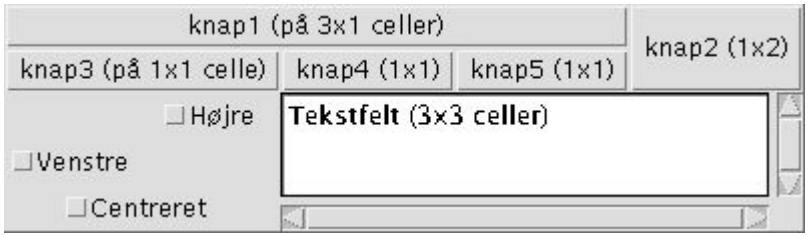
FlowLayout



BorderLayout



GridBagLayout



BoxLayout

intet layout (null)

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
- 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

Kommunikere med databasen

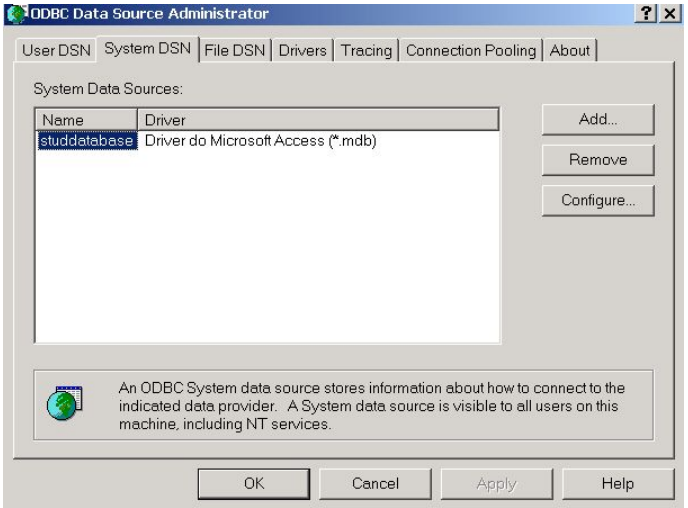
```
import java.sql.*;
public class Databasekommunikation {
  public static void main(String[] arg) throws Exception {
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    Connection forb = DriverManager.getConnection("jdbc:odbc:datakilde1");
    Statement stmt = forb.createStatement();

    // forsøg at slette tabel - hvis den ikke findes opstår en fejl som fanges
    try { stmt.executeUpdate("drop table KUNDER"); } catch (Exception e) {}

    stmt.executeUpdate("create table KUNDER (NAVN varchar(32), KREDIT number)");

    stmt.executeUpdate("insert into KUNDER values('Jacob', -1799)");

    // indsæt data fra variabler
    String navn = "Hans";
    double kredit = 500;
    stmt.executeUpdate("insert into KUNDER(NAVN,KREDIT) values('"+navn+"', "+kredit+")");

    // forespørgsler
    ResultSet rs = stmt.executeQuery("select NAVN, KREDIT from KUNDER");
    while (rs.next())
    {
      navn = rs.getString("NAVN");
      kredit = rs.getDouble("KREDIT");
      System.out.println(navn+" "+kredit);
    }
  }
}
```

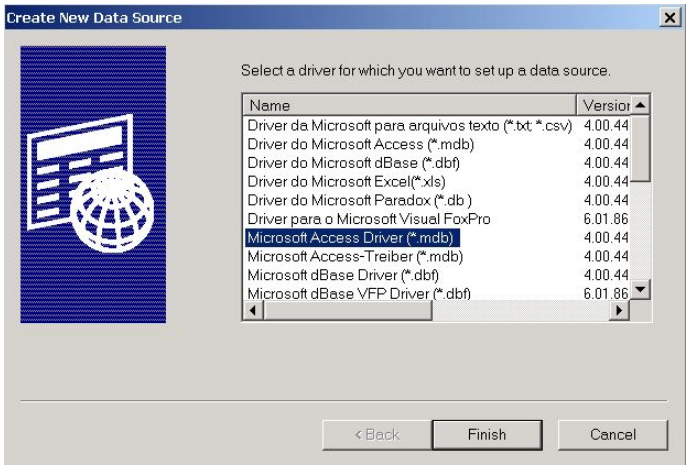
```
Jacob -1799.0
Brian 0.0
Hans 500.0
```

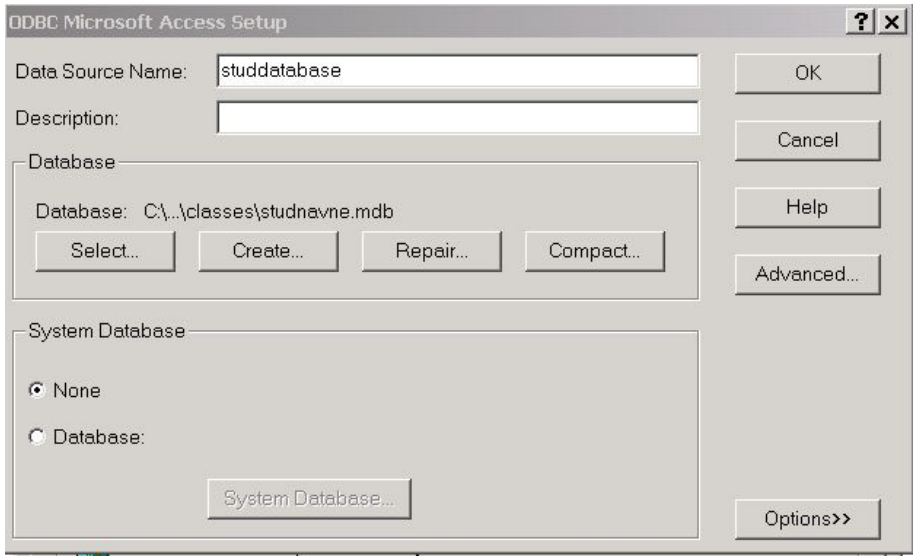
Lave JDBC-ODBC-bro til Access-fil

Eksempel:

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

5. 

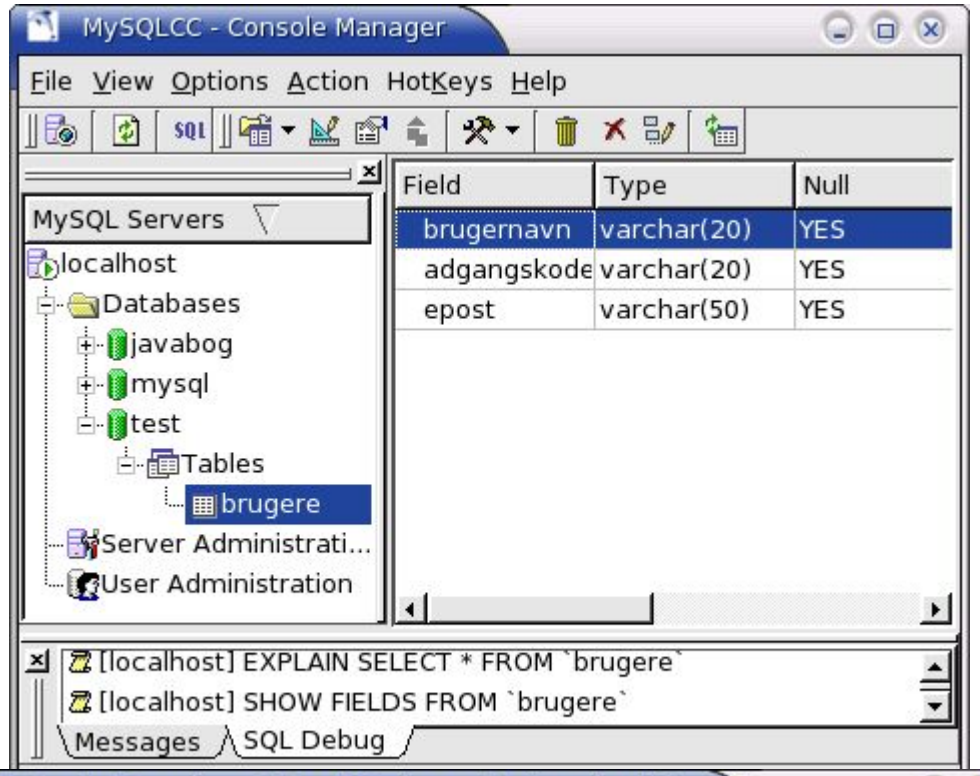
6. 

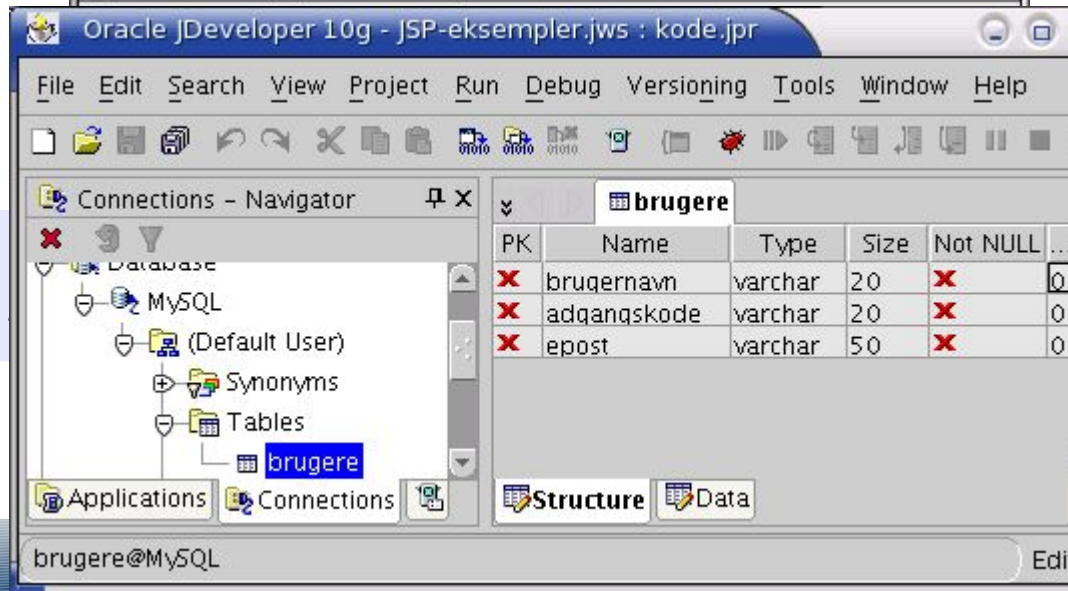
7. 

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();
    }
}
```

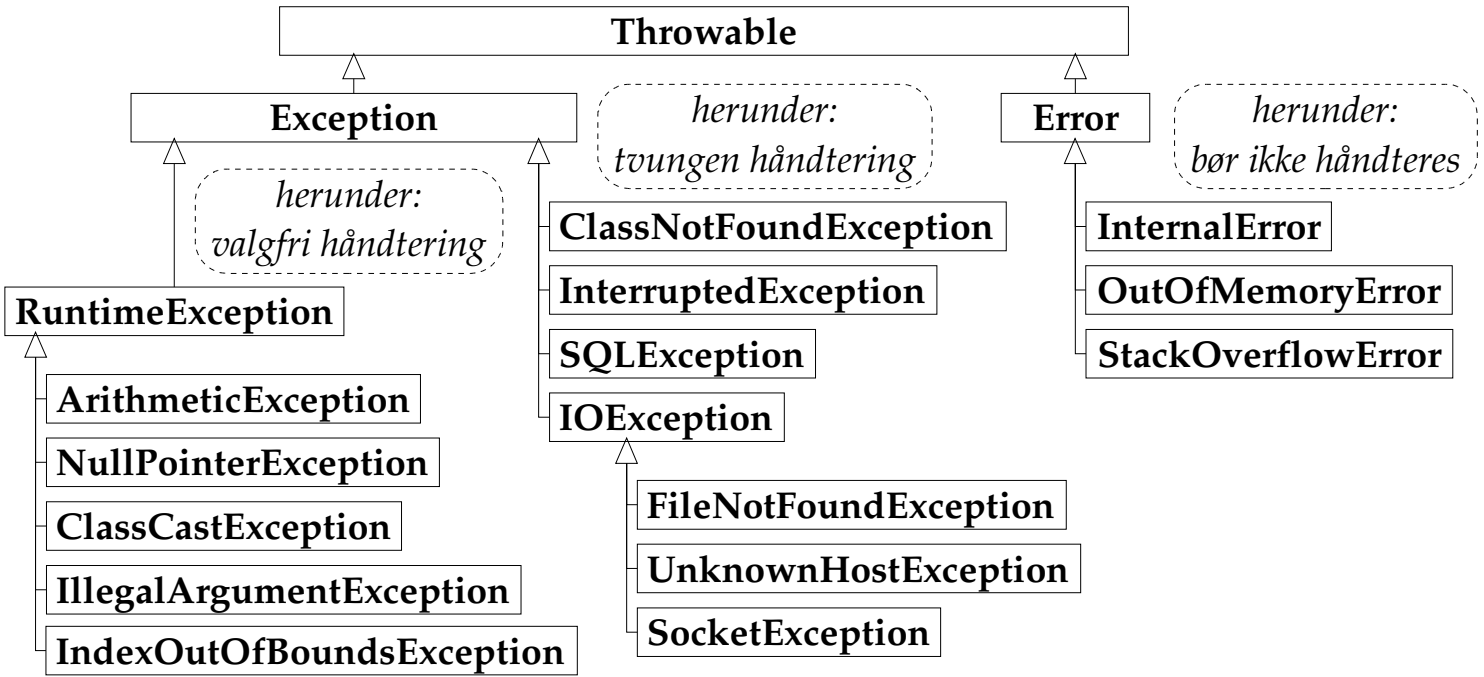
Uden automatisk commit

```
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);
      }
    }
}
```

Undtagelser og stakspor

```
public class SimpelUndtagelse
{
   public static void main(String[] arg)
   {
      System.out.println("Punkt A");       // punkt A
      ArrayList l = new ArrayList();
      System.out.println("Punkt B");       // punkt B
      l.get(5);
      System.out.println("Punkt C");       // punkt C
   }
}
```

```
Punkt A
Punkt B
java.lang.ArrayIndexOutOfBoundsException: 5 >= 0
   at java.util.ArrayList.get(ArrayList.java:417)
   at SimpelUndtagelse.main(SimpelUndtagelse.java:10)
Exception in thread "
```



Undtagelser

- Programudførelsen afbrydes, når der opstår en undtagelse
- Kode, hvori der kan opstå en undtagelse og efterfølgende afhængig kode, bør være i samme try-catch-blok
- Undtagelser med tvungen håndtering skal enten fanges (med try-catch i metodekroppen) eller sendes videre til kalderen (med throws i metodehovedet)

```
try
{
   ...                      // programkode hvor der er en risiko
   ...                      // for at en undtagelse opstår
}
catch (Undtagelsestype u) // Undtagelsestype er f.eks. Exception
{
   ...                      // kode som håndterer fejl af
   ...                      // typen Undtagelsestype
}
...                        // dette udføres både hvis ingen undtagelse opstod
...                        // og hvis der opstod fejl af typen Undtagelsestype
```

Rydde op med finally

- Finally-blokke bliver altid udført, selv når undtagelsen ikke fanges, eller der hoppes ud af metoden på anden vis

```
public void metode1()
{
   try
   {
      ...
      if (...) return;
      if (...) throw new IllegalArgumentException(...);
      ...
   }
   catch (NullPointerException e)
   {
      System.out.println("Intern fejl:");
      e.printStackTrace();
   }
   finally
   {
      System.out.println("Dette bliver altid udført.");
   }
   System.out.println("Slut på metode1()");
}
```

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");
```

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<=antalKolon
          System.out.println();
       }
       System.out.println(" ---------
    }

    public static void main(String[]
    {
       Class.forName("oracle.jdbc.dri
       Connection forb = DriverManage
                 "jdbc:oracle:thin:@

       DatabaseMetaData dmd = forb.ge
       System.out.println("DatabasePr
       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");
```

```
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("");
```

```
<?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>
```