Videregående programmering i Java

Dag 3 - Skabende designmønstre

Skabende designmønstre:
Fabrikeringsmetode/Fabrik, Singleton, Abstrakt fabrik (Toolkit), Prototype, Objektpulje
Eksempel: Forskellige slags dataforbindelser
Objektorienteret analyse
JDBC - databasekommunikation

Læsning: VP 8-8.2, VP 16, analysedel i kapitel 22 af http://javabog.dk (udleveret)

Kursusopgaver - status

Faser i programudvikling

1) Kravene til programmet bliver afdækket.
2) Analyse - **hvad** det er for ting og begreber, programmet handler om.
3) Design - **hvordan** programmet skal fungere.
4) Programmering.
5) Afprøvning (test).

- Forskellige metoder har vidt forskelligt tidsforbrug og antal gentagelser af faserne!
  - (Uigennemtænkt) programmering (3 min, gentag uendeligt)
  - Vandfaldsmodellen (3 måneder, gør kun én gang)
  - UP (Unified Process)
  - Adræt programudvikling, f.eks. XP (Ekstremprogrammering)

Faser i programudvikling

- OBS! Det følgende er *ikke* under Åben Dokumentlicens

Software Engineering

- Software engineering is the systematic process of transforming requirements into high quality software, on time and at cost.

Software Engineering

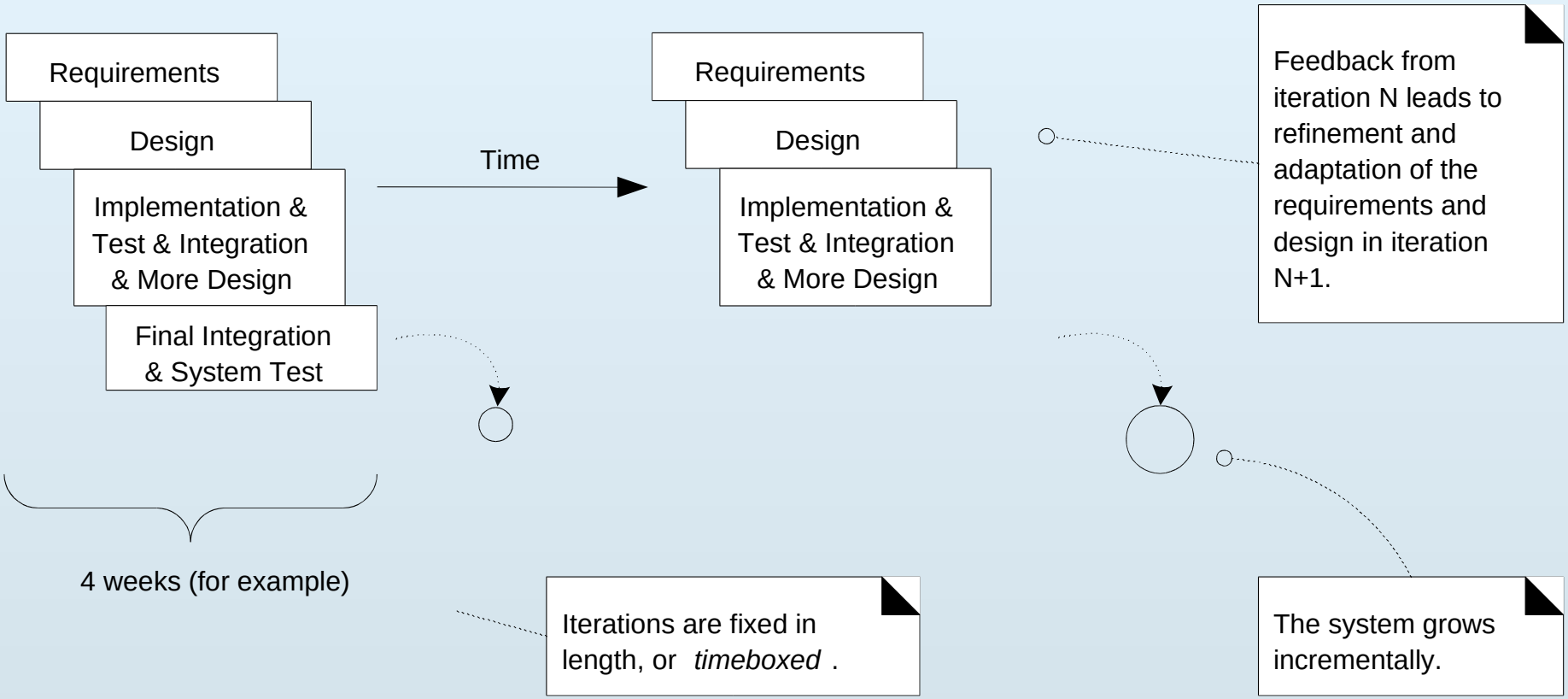
- The software development process involves:

- Requirements
  - Capturing what the system should do.
- Analysis
  - Refining and structuring the requirements.
- Design
  - Realizing the requirements in system architecture.
- Implementation
  - Building the software.
- Test
  - Verifying that the implementation works.

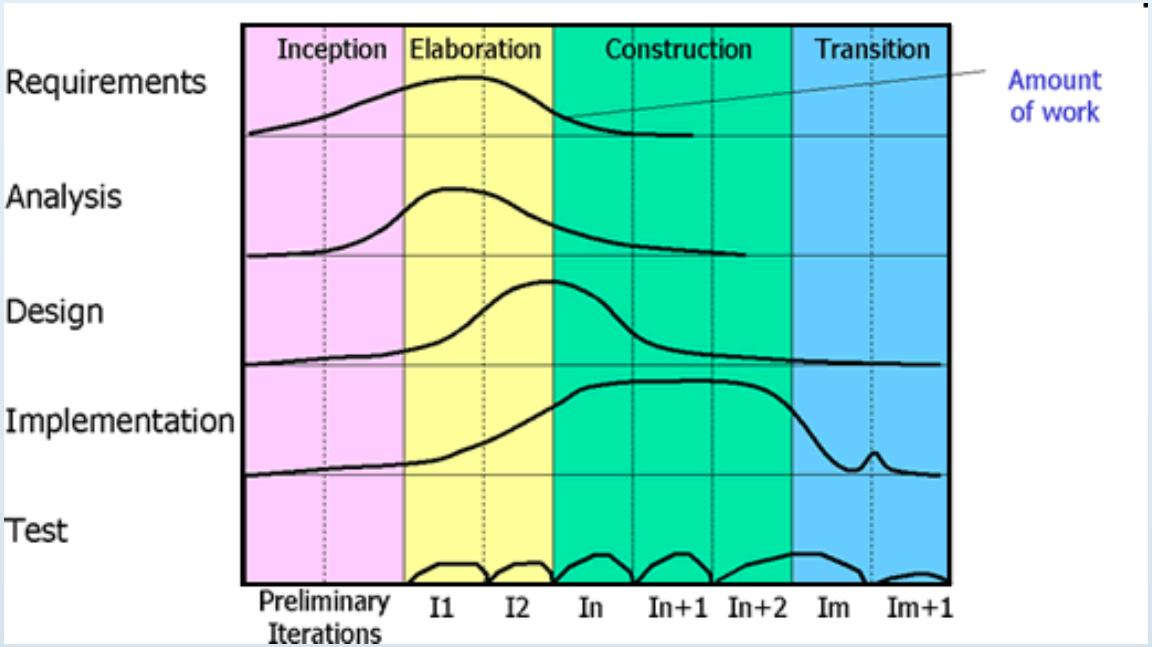
Iterative and Incremental Process

- Software development is an iterative process.
  - The requirements, analysis, design, implementation and test steps are repeated several times during the project.
- Software development is an incremental process.
  - After each iteration, another part of the program is completed. The program gets larger, is incremented.

Iterative and Incremental Process



Unified Process



Analysefase

Hvad det er for ting og begreber, programmet handler om

Redskaber til objektorienteret analyse

- Skrive vigtige ord op
- Brugssituationer (eng.: Use Case)
- Aktivitetsdiagrammer, systemsekvensdiagrammer o.lign.
- Skærmbilleder

Vi bruger nu 10 minutter til hvert punkt!

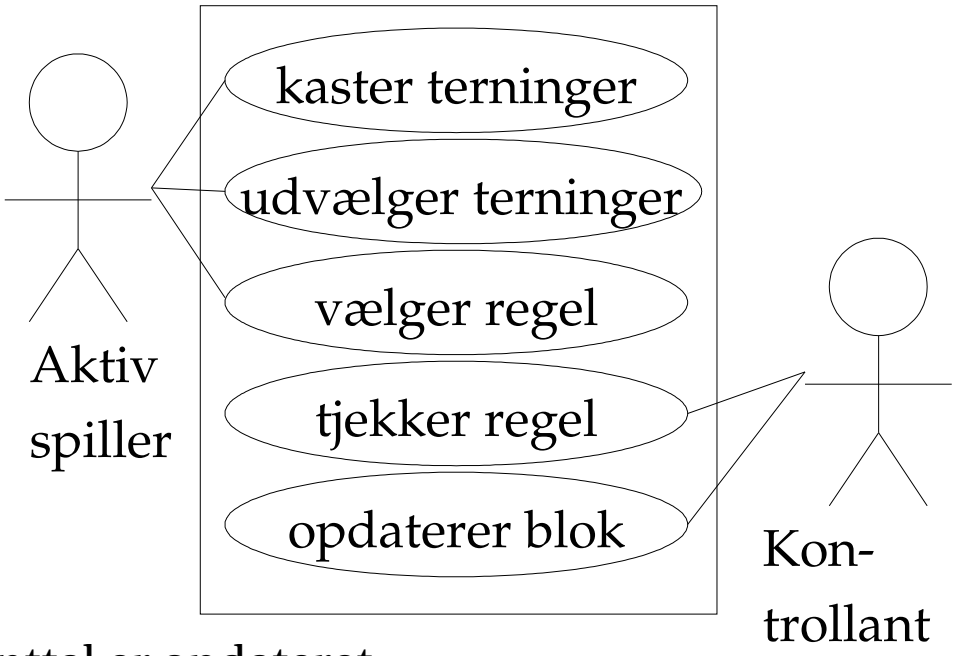
Fremlæggelse næste gang!

Skrive vigtige ord op

- Skriv alle de navneord (i ental) eller ting op, man kan komme i tanke om ved problemet.
- Ud for hver ting kan man notere eventuelle egenskaber (ofte tillægsord) og handlinger (ofte udsagnsord), der knytter sig til tingen.
  - Yatzyspil- antal spillere
  - Terning - værdi, kaste, holde
  - Raflebæger - kombination, ryste, holde
  - Blok - skrive spillernavn på, skrive point på
  - Spiller - navn, type (computer/menneske)
  - Computerspiller - strategi (dum/tilfældig, grådig, strategisk)
  - Menneskespiller
  - Regel (kunne også kaldes en mulighed eller et kriterium) - opfyldt, brugt, antal point
  - Lager - hiscore

Brugssituationer (eng.: Use Case)

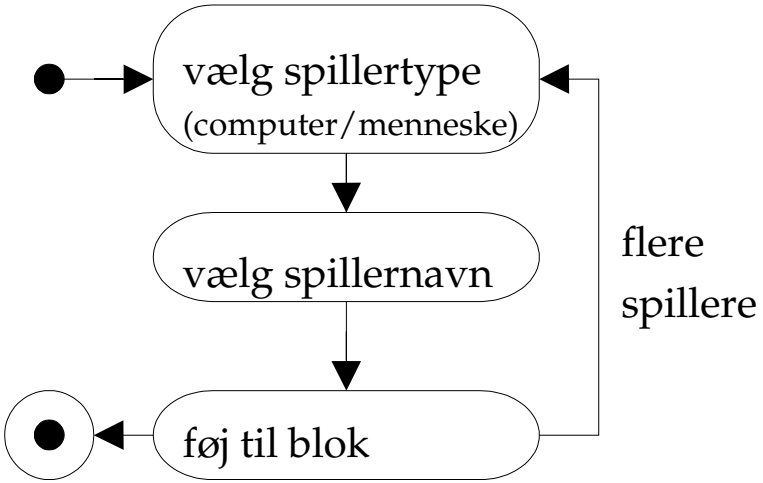
- Som diagrammer ---------->
- På listeform (anbefales):
  - Primær aktør
    Brugeren, hvis tur det er
  - Interessenter
    Systemet
  - Tilstand før
    Det er brugerens tur
  - Tilstand efter
    Bruger har valgt felt i blokken og alle pointtal er opdateret
  - Hovedscenarie
    1. Bruger trykker på "kast terninger"
    2. Terninger, der ikke er holdt får en ny tilfældig værdi
    3. Bruger vælger terninger der skal holdes
       (punkt 1-3 gentages maks. 3 gange)
    4. Systemet viser en liste af mulige felter i blokken
    5. Bruger vælger et felt
    6. ...
  - Alternativer til hovedscenarie
    2a. Alle terninger er holdt: Advarselsvindue dukker op: "Vil du afslutte kastene?"

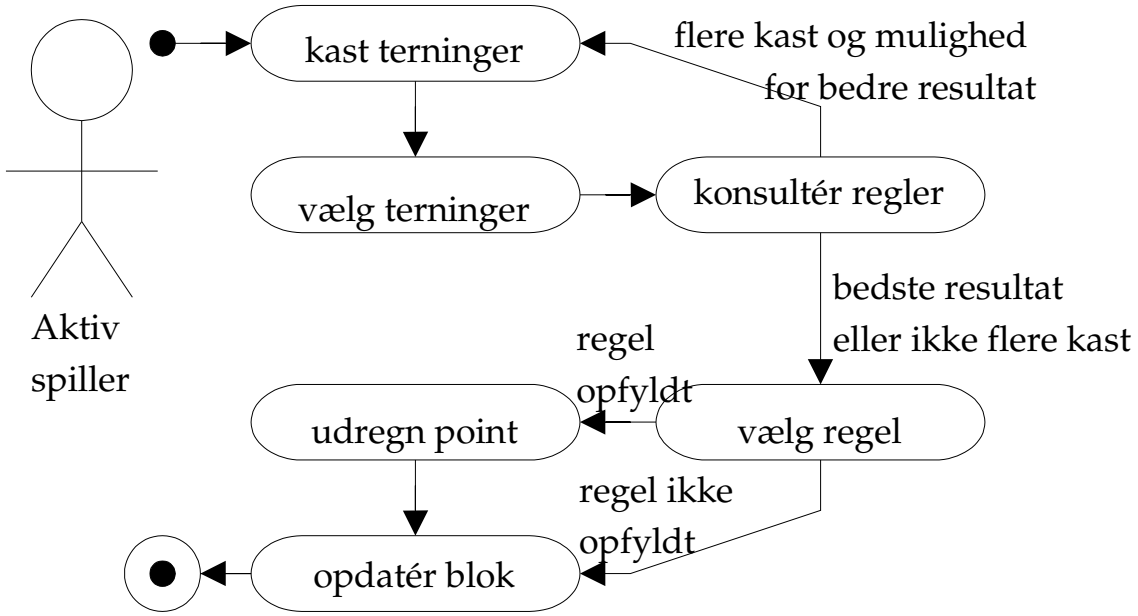


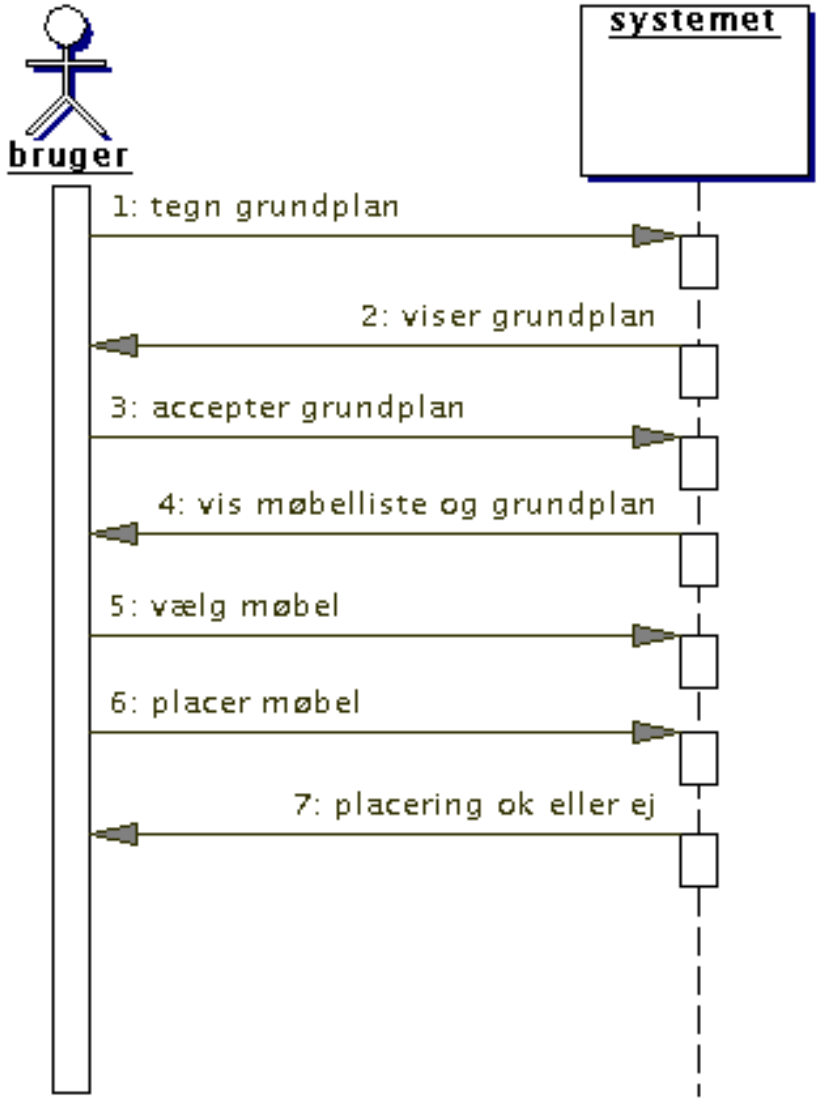
Brugssituationer (eng.: Use Case)

- Skabeloner til listeform
  - http://alistair.cockburn.us/usecases/usecases.html
  - http://usecases.org/

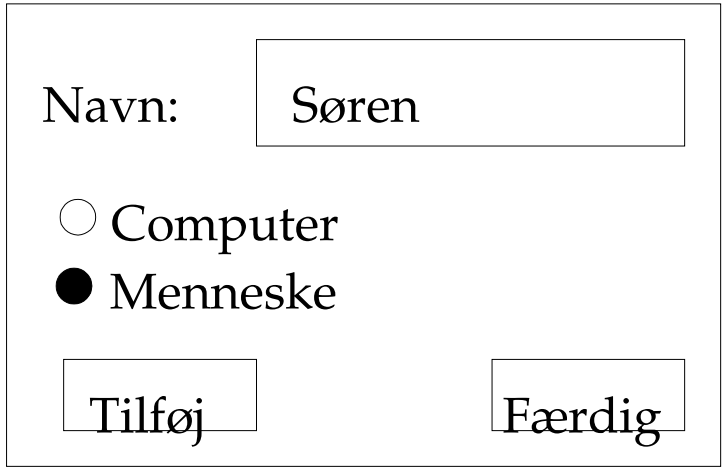
Aktivitetsdiagrammer, systemsekvensdiagrammer o.lign.

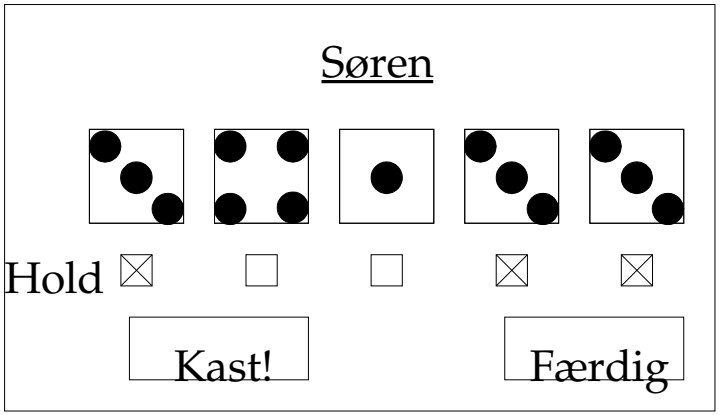






Skærmbilleder





	Jacob	**Søren**
Ettere	4	
Toere		
Treere		9
etc...		
Sum		
Bonus		
Et par		
etc...		
Sum		

Skabende designmønstre

```
// høj kobling - klient opretter et Hjælp-objekt
Hjælp h = new Hjælp();




...
h.metode1();
h.metode2();
```

MEN... det kunne være at:

- Det var en nedarving af Hjælp, der skulle oprettes (polymorfi)
- Objektet skulle oprettet med nogle bestemte parametre i konstruktøren
- Det samme objekt skulle bruges af alle klienter (en Singleton)
- Eksisterende Hjælp-objekter skulle genbruges (en Objektpulje)

Skabende designmønstre

```
// høj kobling - klient opretter et Hjælp-objekt
Hjælp h = new Hjælp();




...
h.metode1();
h.metode2();
```

MEN... det kunne være at:

- Det var en nedarving af Hjælp, der skulle oprettes (polymorfi)
- Objektet skulle oprettet med nogle bestemte parametre i konstruktøren
- Det samme objekt skulle bruges af alle klienter (en Singleton)
- Eksisterende Hjælp-objekter skulle genbruges (en Objektpulje)

- Oprettelse af objekt afgør objektets præcise type!
  - Dette er (for) stærk kobling i visse tilfælde
- Den del af programmet (klienten) som *bruger* visse objekter skal ikke altid også *oprette* disse objekter

Skabende designmønstre

```
// høj kobling - klient opretter et Hjælp-objekt
//Hjælp h = new Hjælp();

// fabrikeringsmetode leverer objekt til klienten
Hjælp h = Hjælp.opretHjælp();

...
h.metode1();
h.metode2();
```

Det kan være at:

- Det er en nedarving af Hjælp, der bliver oprettet (polymorfi)
- Objektet bliver oprettet med nogle bestemte parametre i konstruktøren
- Det samme objekt bliver brugt af alle klienter (en Singleton)
- De eksisterende Hjælp-objekt bliver genbrugt (en Objektpulje)

- Fabrikeringsmetode (eng.: Factory Method)
  - En metode, der opretter et objekt for klienten
  - Afkobler (mindsker graden af bindinger) mellem
    - oprettelsen af nogle bestemte objekter (i ét modul)
    - anvendelsen af dem af (i et andet modul, klienten)

Designmønstret Fabrik

(eng.: Factory)

(objekt med fabrikeringsmetode)

Problem: Klienten kan/skal ikke bestemme præcist, hvordan nogle objekter oprettes.

Løsning: Lad en Fabrik med en fabrikeringsmetode varetage oprettelsen.

```
// høj kobling - klient opretter et Hjælp-objekt
Image i = new Image("billede.gif");      // forkert!!
```

Image-objekter kan være forskelligt repræsenteret afhængig af type (GIF, JPG eller PNG) og opløsning

```
// fabrikeringsmetode leverer objekt til klienten
// this er et grafisk objekt, f.eks. applet, panel,..
Image i = this.getImage("billede.gif");  // korrekt
```

Designmønstret Singleton

(en klasse, der må være én og kun én instans af)

Problem: Klienten må ikke have flere objekter af en bestemt type, men skal altid bruge det samme objekt.

Løsning: Programmér sådan, at der aldrig kan oprettes mere end ét eksemplar af det pågældende objekt.

Eksempler

java.lang.Runtime (det kørende program)
java.awt.Toolkit (implementationen af grafiksystemet/AWT)

```
Runtime rt = Runtime.getRuntime();
// eksempler på brug af Runtime-objektet
System.out.println("Hukommelse reserveret til Java: "+rt.totalMemory());
System.out.println("Heraf ledigt: "+rt.freeMemory());
rt.gc(); // kør garbage collector
System.out.println("Nu ledigt: "+rt.freeMemory());
```

```
Toolkit tk = Toolkit.getDefaultToolkit();
// eksempler på brug af Runtime-objektet
System.out.println("Skærmstørrelse (punkter): " + tk.getScreenSize());
tk.beep(); // computeren siger bip
Image i = tk.getImage("billede.gif"); // her fungerer Toolkit som fabrik
```

Implementering af Singleton

- Normal implementering
  - Privat konstruktør
  - Instans oprettes ved klasseindlæsning og gemmes i privat klassevariabel
  - Fabrikeringsmetode returnerer den private instans

```
public class Dataforbindelse
{
    private static Dataforbindelse instans = new Dataforbindelse();

    public static Dataforbindelse hentForbindelse() { return instans; }

    private List alle;
    private Dataforbindelse() { alle = new ArrayList(); }

    public void sletAlleData() { alle.clear(); }
    public void indsæt(Kunde k) { alle.add(k); }
    public List hentAlle() { return alle; }
}
```

```
Dataforbindelse1 dbf = Dataforbindelse.hentForbindelse();
dbf.indsæt( new Kunde("Kurt",1000) );
```

Nedarvinger af Dataforbindelse?

Implementering af Singleton

- Normal implementering
  - Privat konstruktør
  - Instans oprettes ved klasseindlæsning og gemmes i privat klassevariabel
  - Fabrikerings(klasse)metode returnerer den private instans

- Andre implementeringer
  - Instans oprettes først når fabrikeringsmetode kaldes første gang
    - Fabrikeringsmetode må tjekke om instans allerede er oprettet
      - Trådsikkerhed kan blive et problem - fabrikeringsmetode skal være synchronized
  - Ingen fabrikeringsmetode, public final klassevariabel med instans
  - Ikke-privat konstruktør
    - Tillader nedarving
    - Konstruktør må tjekke om instans allerede er oprettet
      - Kast undtagelse hvis instans allerede findes
      - Trådsikkerhed kan blive et problem - konstruktør skal være synchronized
  - Fabrikeringsmetode i anden klasse
    - Konstruktør med pakke-synlighed

Andre implementeringer af Singleton

```
// Instans oprettes først når fabrikeringsmetode kaldes første gang
public class Dataforbindelse
{
    private static Dataforbindelse instans = null;

    public static synchronized Dataforbindelse hentForbindelse() {
       if (instans == null) instans = new Dataforbindelse();
       return instans;
    }
```

```
// Ingen fabrikeringsmetode, public final klassevariabel med instans
public class Dataforbindelse
{
    public static final Dataforbindelse instans = new Dataforbindelse()
```

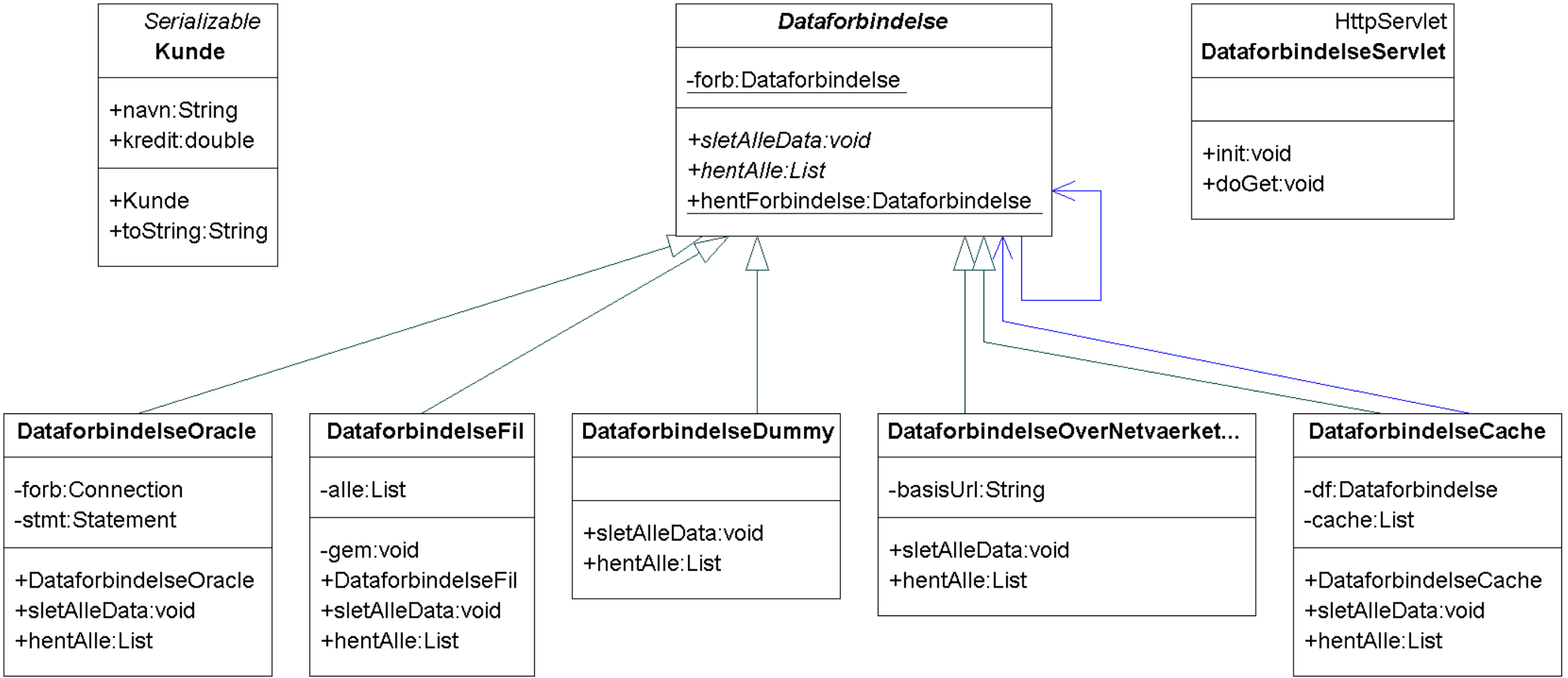
```
// Ikke-privat konstruktør
public class Dataforbindelse
{
    public static Dataforbindelse instans = null;

    public static synchronized Dataforbindelse hentForbindelse() {
       if (instans == null) instans = new Dataforbindelse();
       return instans;
    }

    protected Dataforbindelse() {
       if (instans != null) throw new IllegalAccessException("Obj findes");
       instans = this;
```

Eksempel: Dataforbindelse

- Indkapsl datalagring i klasse
- Hvis man ønsker fleksibilitet omkring hvor data lagres
  - Start: DataforbindelseDummy, DataforbindelseFil
  - Slut: DataforbindelseOracle, ...



Designmønster
Abstrakt Fabrik / Toolkit

(Fabrik med abstrakt superklasse og nedarvinger, som tager sig af oprettelsen)

Problem: En Fabrik bliver uforholdsmæssigt kompliceret, fordi nogle ydre omstændigheder har stor indflydelse på, hvordan oprettelsen skal foregå.

Løsning: Lav en Abstrakt Fabrik (eng.: Abstract Factory) med en nedarving (Fabrik) for hver omstændighed.

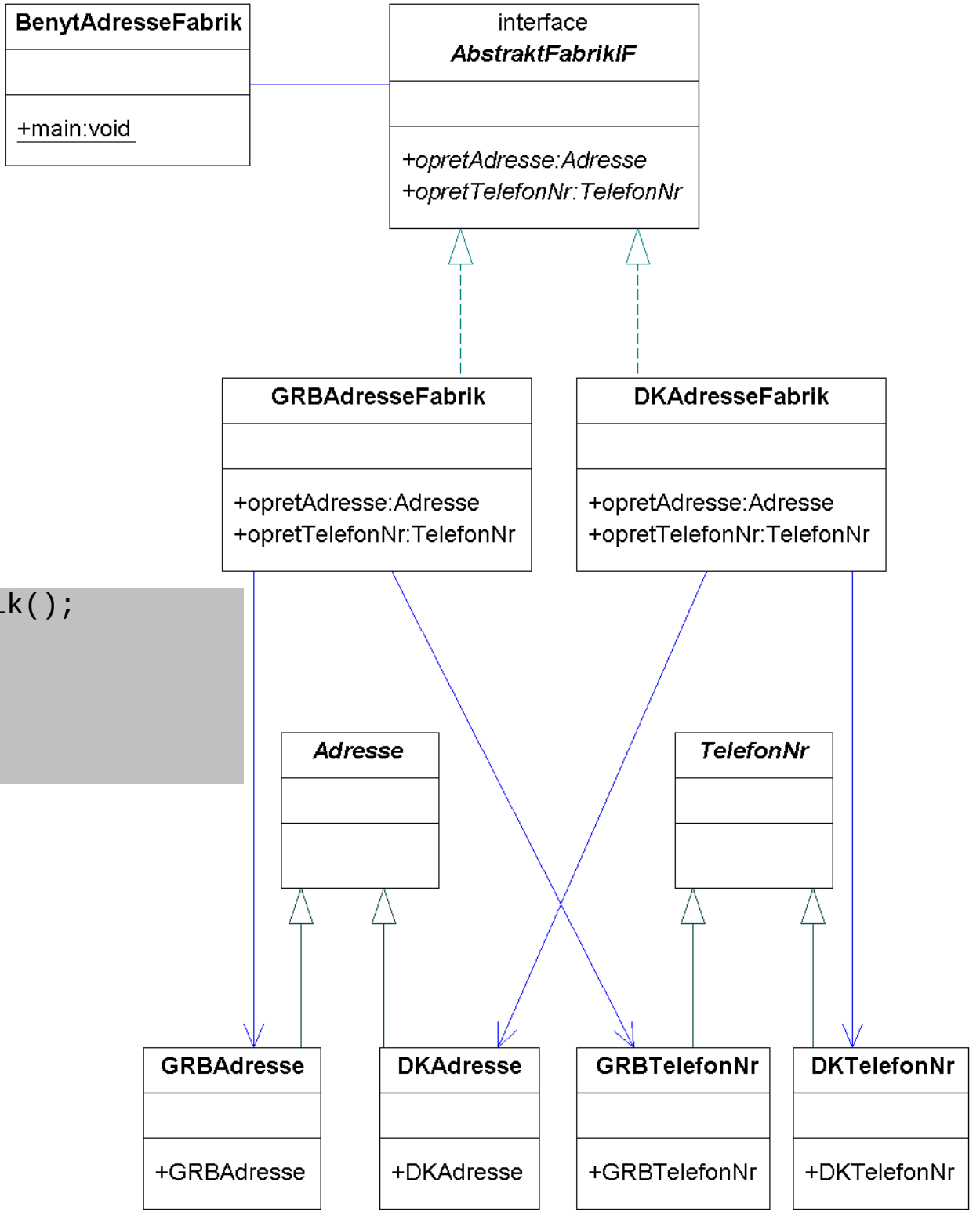
Eksempel: java.awt.Toolkit

Fabrikerer platformsspecifik del af AWT-komponent (peer)

Nedarvinger: WindowsToolkit, LinuxToolkit, SolarisToolkit

```
// følgende gøres i f.eks. java.awt.Button (aldrig fra normalt program!)
package java.awt;
public class Button extends Component {

   public Button() {
     Toolkit tk = Toolkit.getDefaultToolkit();
     ButtonPeer peer = tk.createButton(this); // platformsspecifik del!
```




```
AbstraktFabrikIF af = hentAdresseFabrik();

Adresse a = af.opretAdresse();
TelefonNr tlf = af.opretTelefonNr();
...
```

Designmønstret Prototype

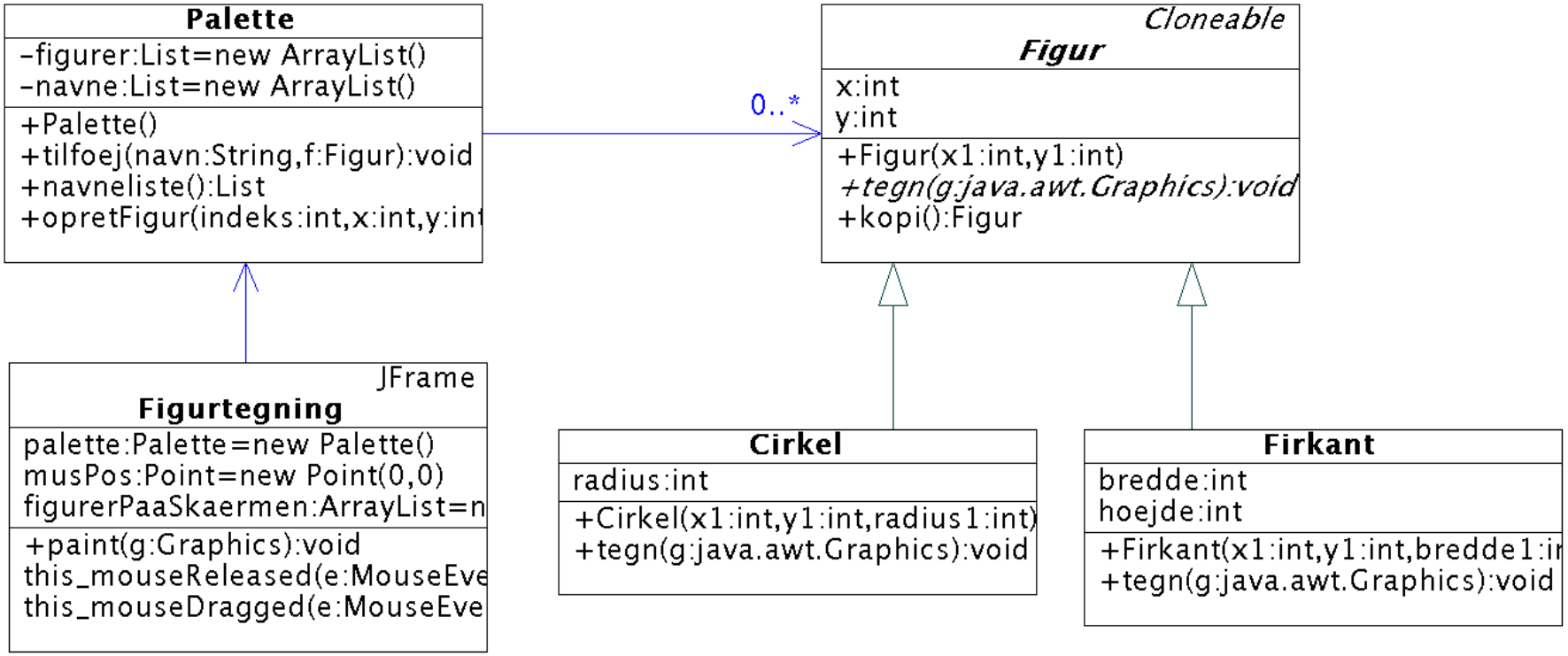
(objekter oprettes ud fra en skabelon)

Problem: Klienten ved ikke, hvad der skal oprettes, men kan dog angive et andet objekt, som ligner det, der skal oprettes

Løsning: Brug det andet objekt som Prototype, og opret objektet ud fra prototypen

Prototyper i et tegneprogram

- Palette har liste af figur-prototyper
  - Liste kan senere nemt udvides
- Bruger kan vælge i listen
  - Når der vælges i paletten, anvendes det pågældende element som Prototype til objektet, der skal tegnes på skærmen



Designmønstret Objektpulje

Problem: Der er et begrænset antal resurser, som skal deles.

Problem: Der oprettes for mange objekter. Programmet er langsomt eller kører ujævnt, fordi der oprettes så mange objekter, der løbende smides væk igen.

Objekterne kunne egentligt godt genbruges i stedet for at blive smidt væk, men oprettelsen sker spredt rundt i programmet, så det er svært at koordinere.

Løsning: Lad et objekt varetage resurserne/objekterne. Lad klienter reservere og frigive objekter gennem dette objekt.

```
public class Objektpulje
{
    private ArrayList ledige = new ArrayList();

    public synchronized void sætInd(Object obj) { ledige.add(obj); }

    public synchronized Object tagUd() {
       if (ledige.isEmpty()) throw new RuntimeException("Ikke flere objekter!");
       Object obj = ledige.remove(ledige.size()-1); // tag objekt ud af puljen
       return obj;
    }
}
```

Designmønstret Objektpulje

Andre muligheder hvis puljen løber tør for objekter

- Lad puljen oprette nye objekter (evt. v.hj.a. en Fabrik): Øvelse
- Lad klient 'hænge' og vente på at et objekt bliver ledigt:

```
public class ObjektpuljeKlientHaenger
{
    private ArrayList ledige = new ArrayList();

    public synchronized void sætInd(Object obj) {
       ledige.add(obj);
       this.notify(); // væk eventuelle ventende tråde
    }

    public synchronized Object tagUd() {
       try {
          while (ledige.isEmpty()) // så længe der ikke er ledige objekter...
          {
             System.out.println("Ikke flere objekter i puljen, venter...");
             this.wait();           // .... vent på at blive vækket
          }
          Object obj = ledige.remove(ledige.size()-1); // tag objekt ud af puljen
          return obj;
       } catch (InterruptedException e) { return null; }
    }
}
```

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:

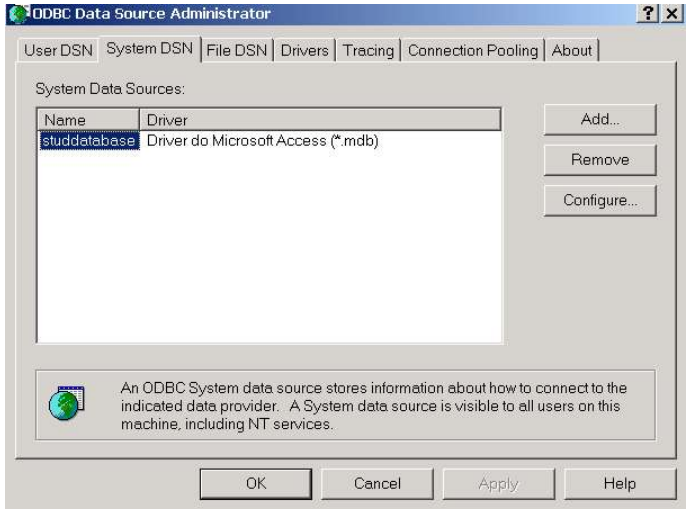
- Type 1: JDBC-ODBC-broen. Langsomste og kun til Windows.
- Type 2: Drivere skrevet i C eller C++ til den specifikke platform (normalt de hurtigste).
- Type 3: Platformsuafhængig (ren Java-) driver med databaseuafhængig kommunikationsprotokol
- Type 4: Platformsuafhængig (ren Java-) driver skrevet til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

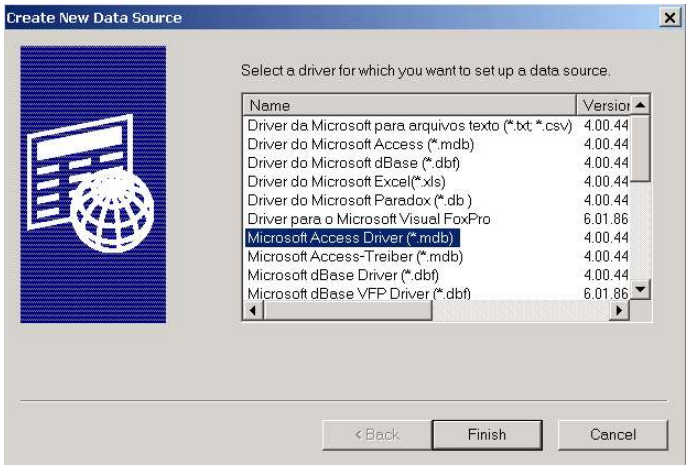
JDBC – databaseadgang

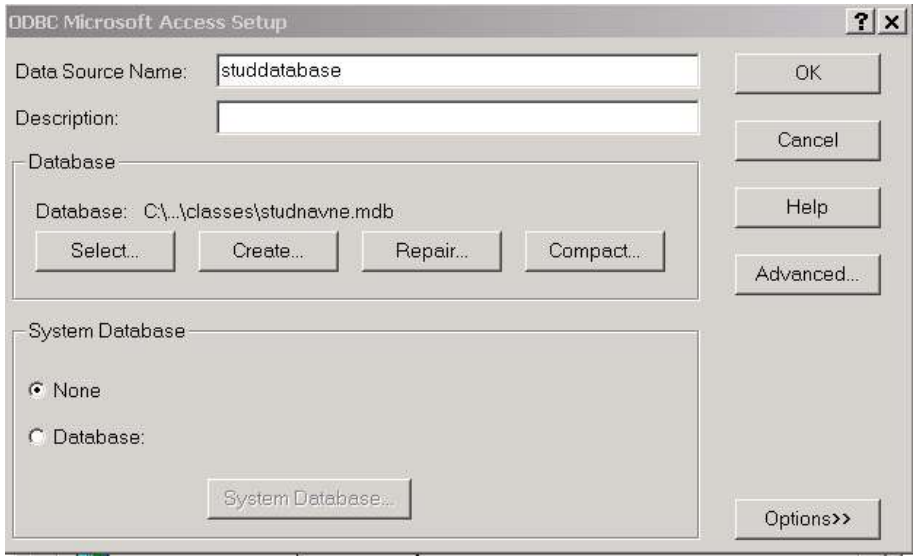
Lave JDBC-ODBC-bro til Access-fil

Eksempel:

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

5. 

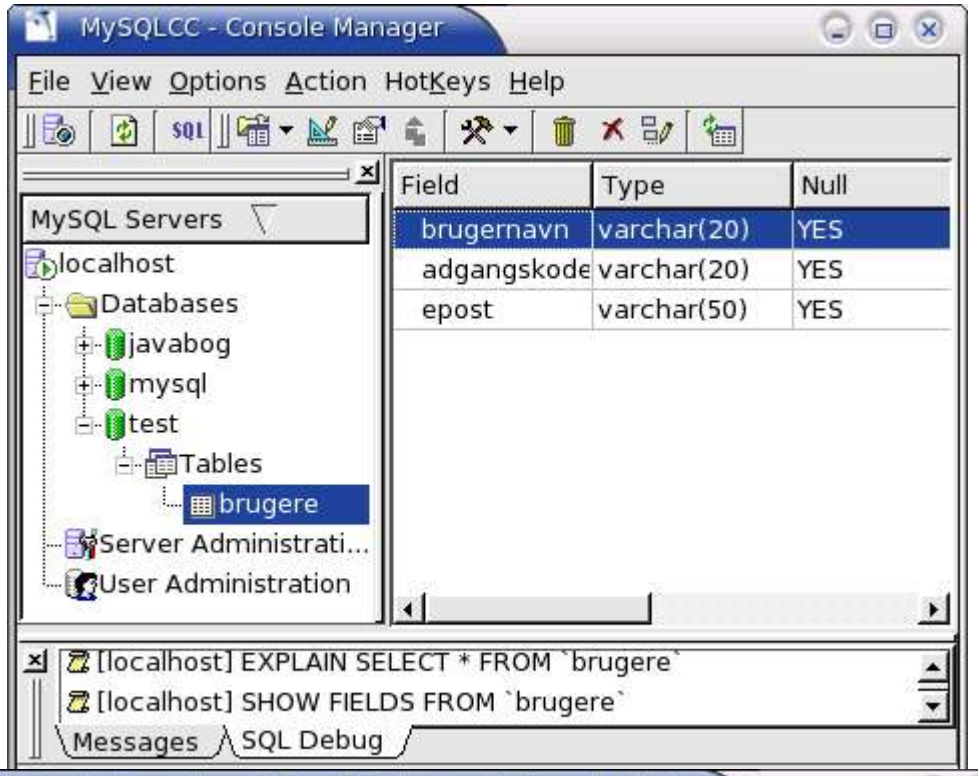
6. 

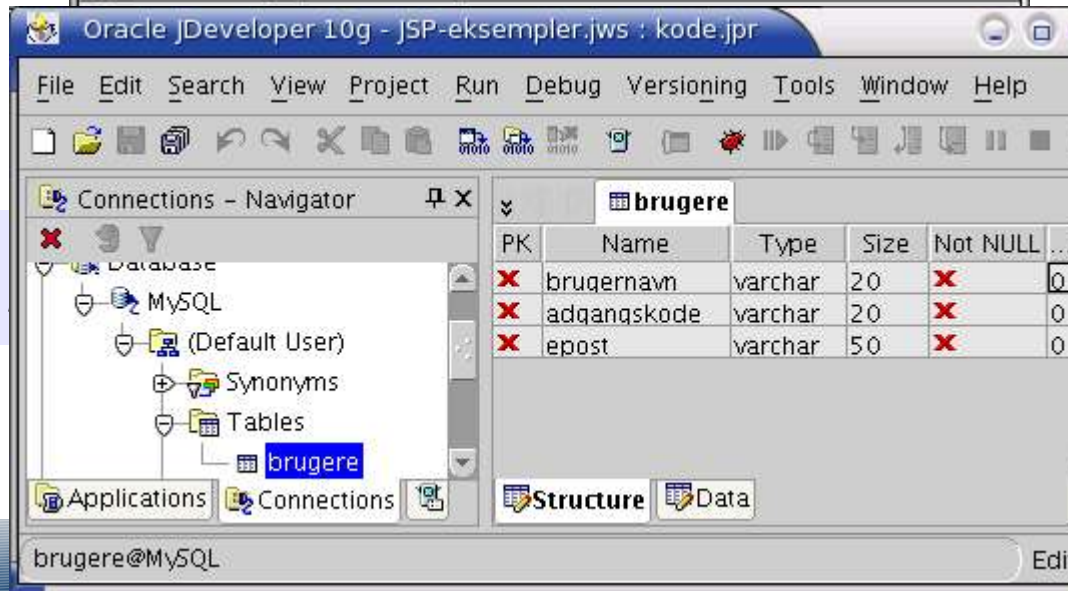
7. 

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();
    }
}
```

JDBC og dens brug af designmønstre