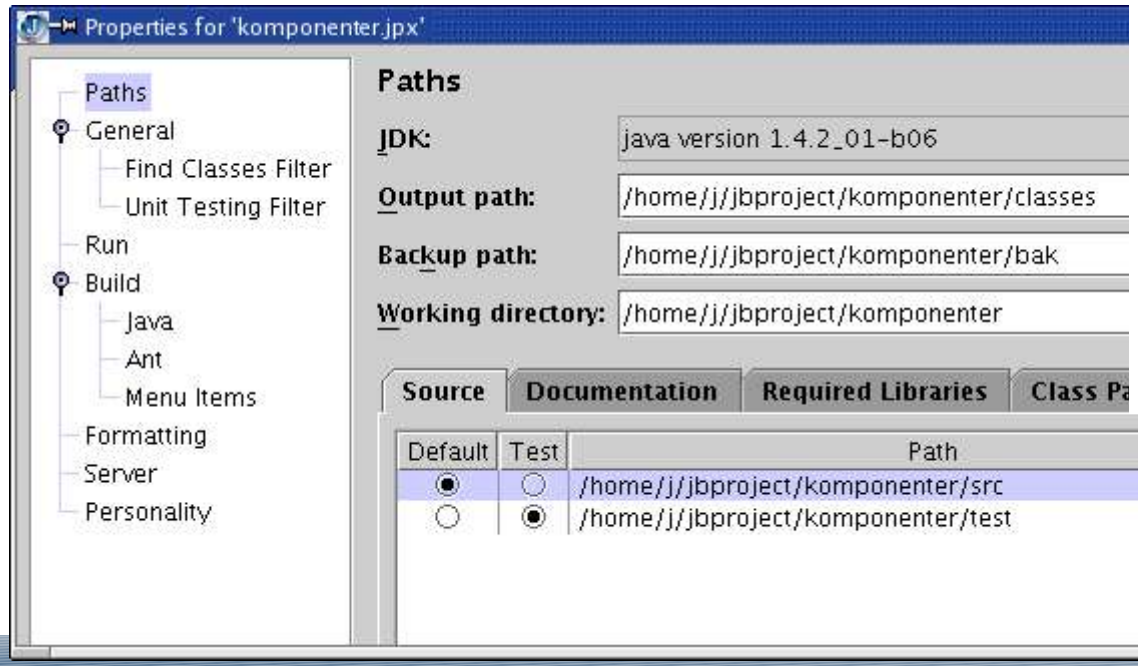
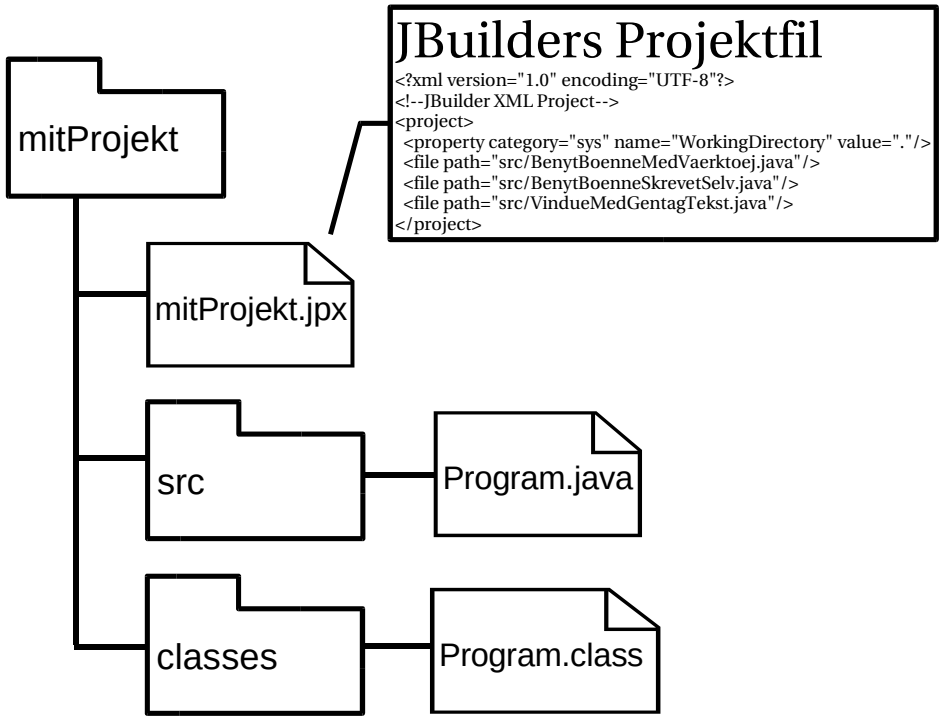
Java-opgraderingskursus

Danmarks Meteorologiske Institut

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





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;
   }
}
```

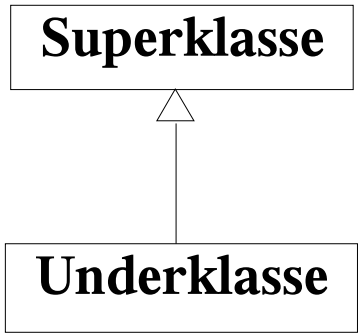
Boks3
-længde :double -bredde :double -højde :double
+Boks3() +Boks3(lgd, b, h) +sætMål(lgd, b, h) +volumen() :double


```
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
```

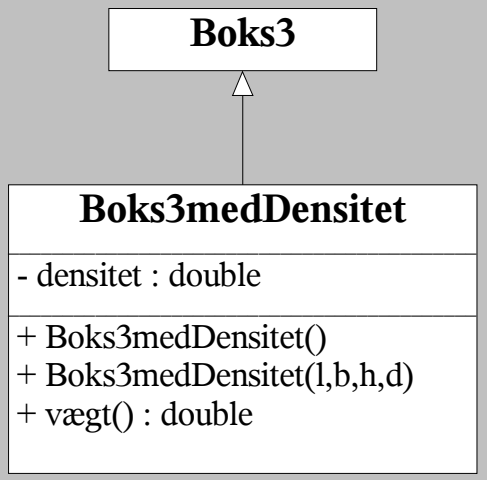
Arv

- 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
    }
}
```



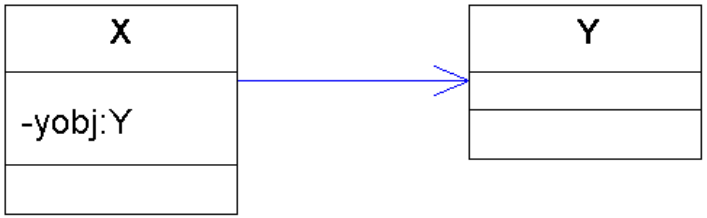
Arv og konstruktører

- 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.

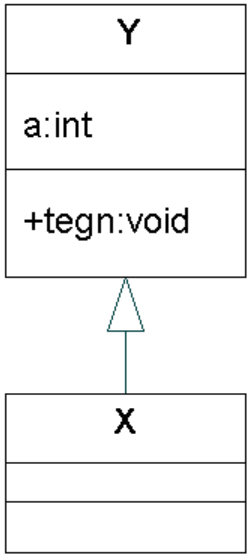


Eksempel:

```
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.

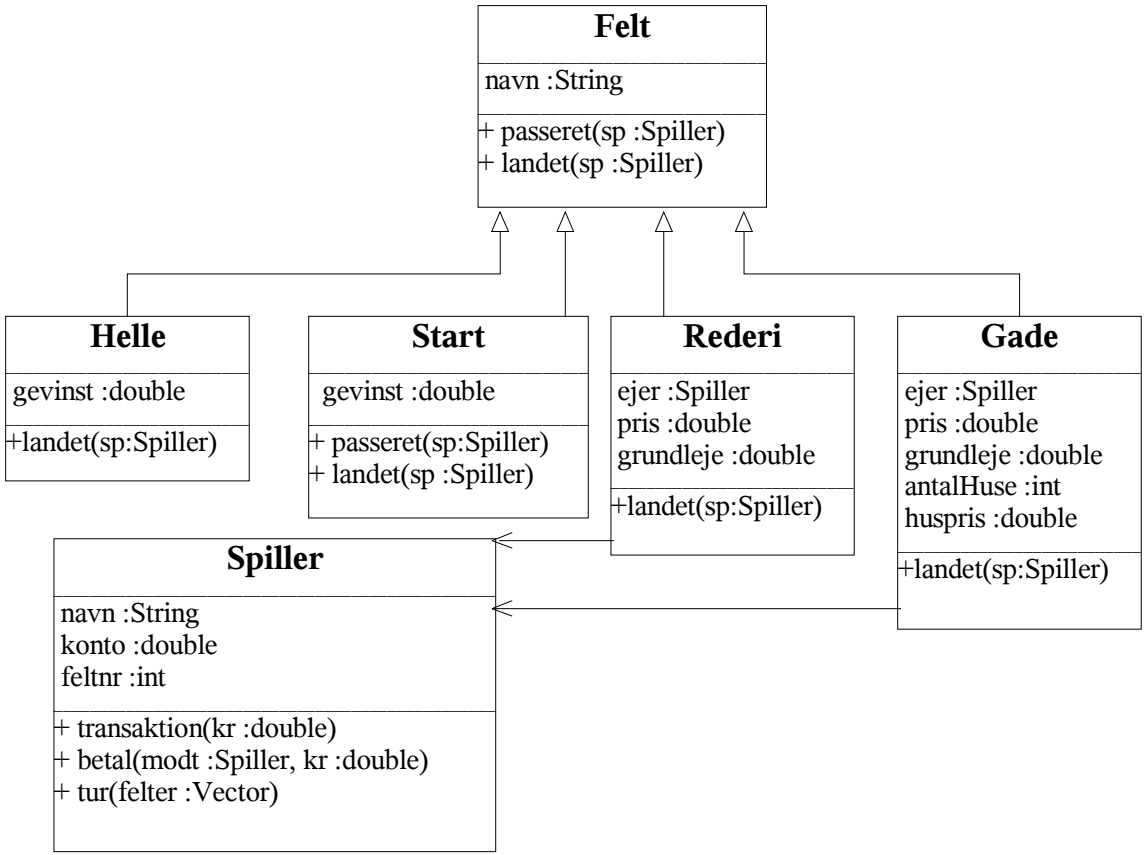


Eksempel:

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

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

UML og klasserelationer

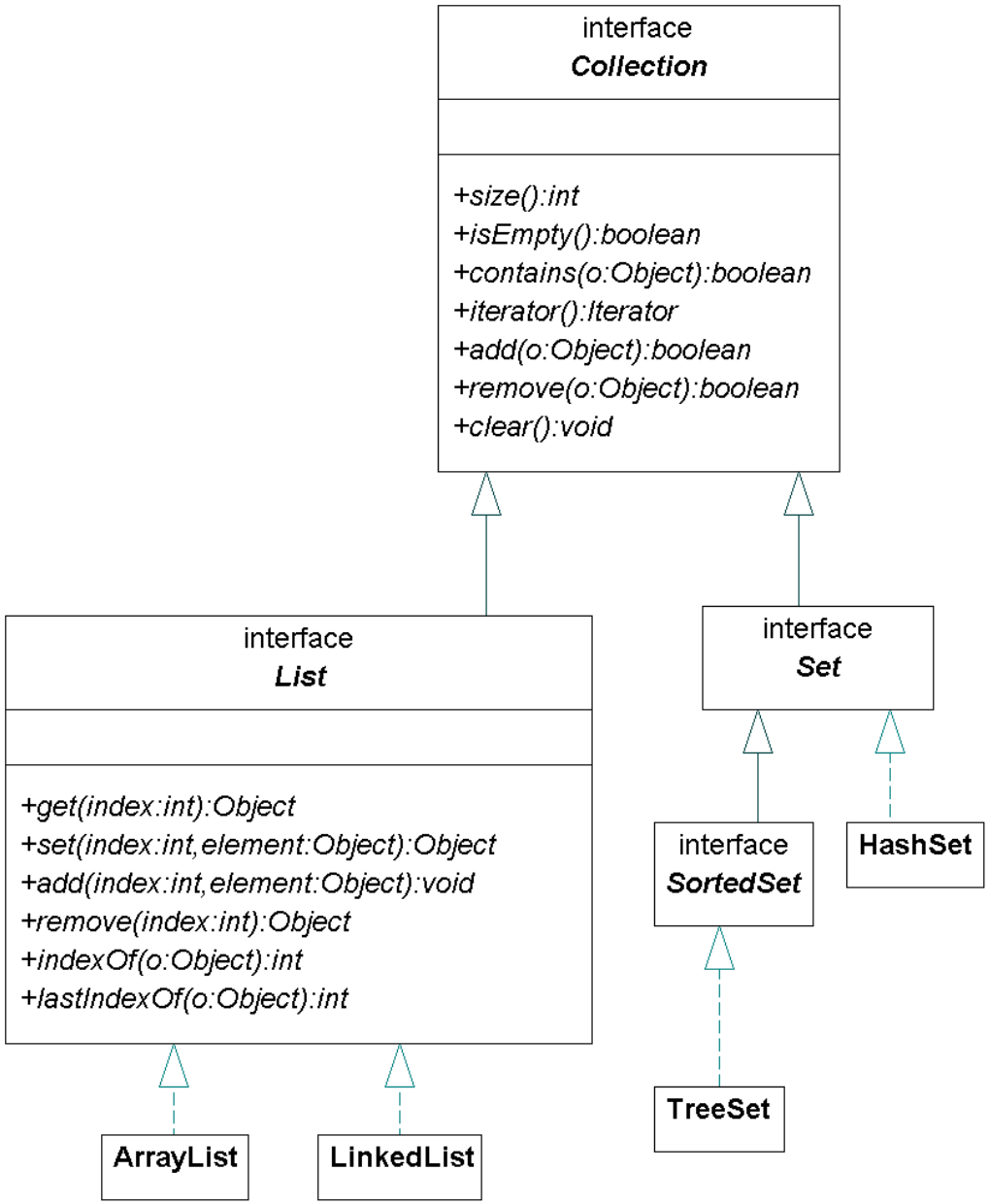


Rederi og Gade *har en* Spiller.

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)

Versioner af JDK

- Problem
  - Produktionsmiljø er JDK 1.3, udviklingsmiljø er JDK 1.5
    - Metoder og klasser brugt i udvikling findes ikke i produktionsmiljø
- Løsninger
  - Opgradere produktionsmiljø
    - Ikke altid muligt (afh. af styresystem, andre projekter, ...)
  - Installere ekstra JDK (f.eks. 1.3) i udviklingsmiljø
    - Hente og installere gammel JDK fra Sun
    - JBuilder understøtter dette (projektegenskaber / Paths / JDK)
    - Webapplikationer: Der skal måske også installeres gammel server
      - Kan være problem i JBuilder

Faser i programudvikling

1) Kravene til programmet bliver afdækket.
2) Analyse - **hvad** det er for ting og begreber, program-
   met 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)
  - XP (Ekstremprogrammering)

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!

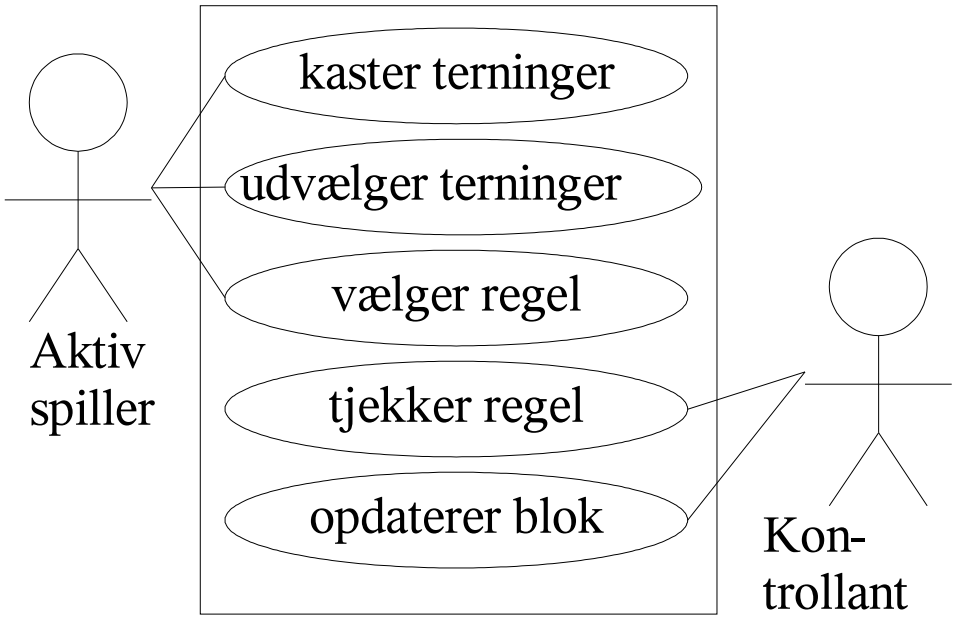
Fremlæggelse næste gang!

Skrive vigtige ord op

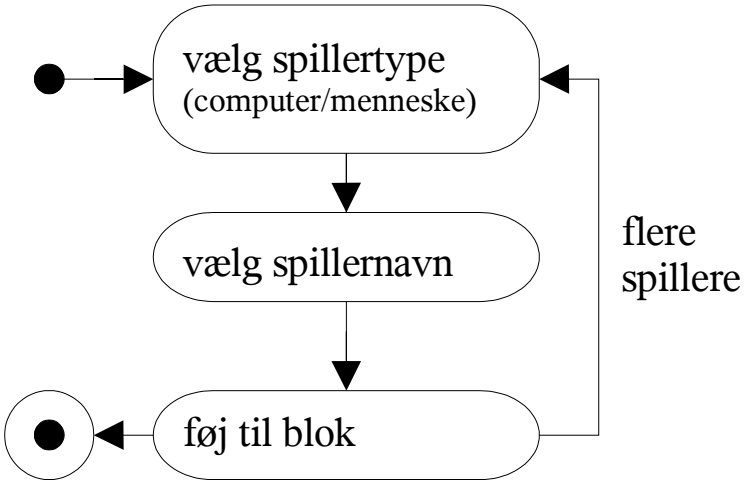
- Skriv alle de navneord (i ental) eller ting op, man kan komme i tanke om ved problemet.
- Ud for hver ting kan man notere eventuelle egenskaber (ofte tillægsord) og handlinger (ofte udsagnsord), der knytter sig til tingen.
  - Yatzyspil- antal spillere
  - Terning - værdi, kaste, holde
  - Raflebæger - 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

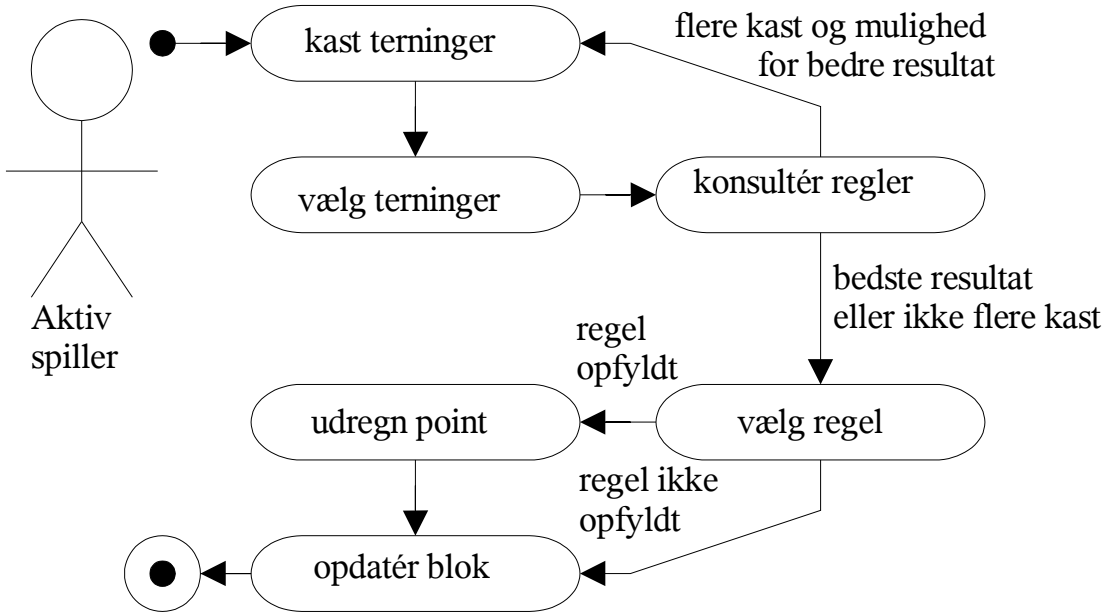
Brugssituationer (eng.: Use Case)

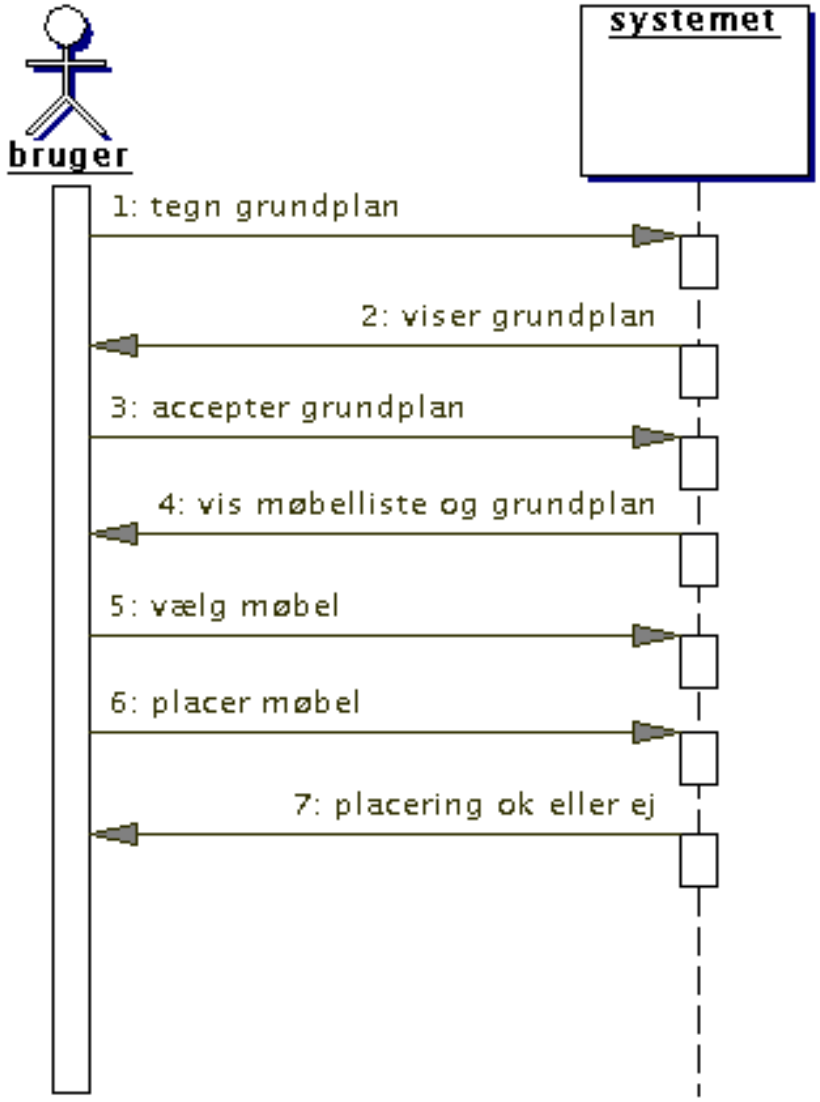
- Som diagrammer ---------->
- På listeform (anbefales):
  - Primær aktør
    Brugeren, hvis tur det er
  - Interessenter
    Systemet
  - Tilstand før
    Det er brugerens tur
  - Tilstand efter
    Bruger har valgt felt i blokken og alle pointtal er opdateret
  - Hovedscenarie
    1. Bruger trykker 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?"



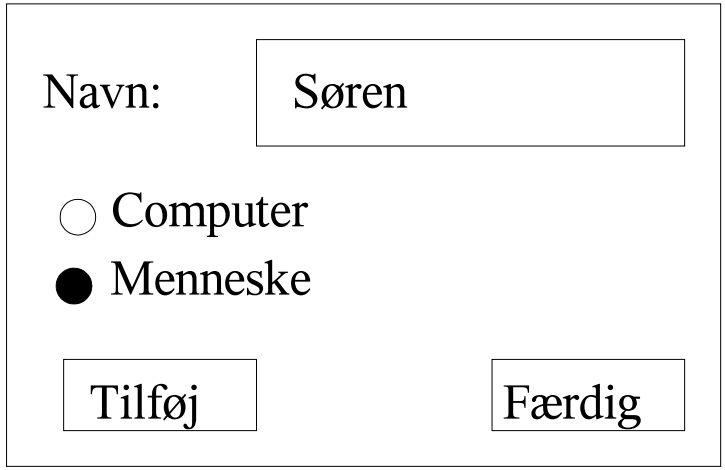
Aktivitetsdiagrammer, systemse-kvensdiagrammer o.lign.

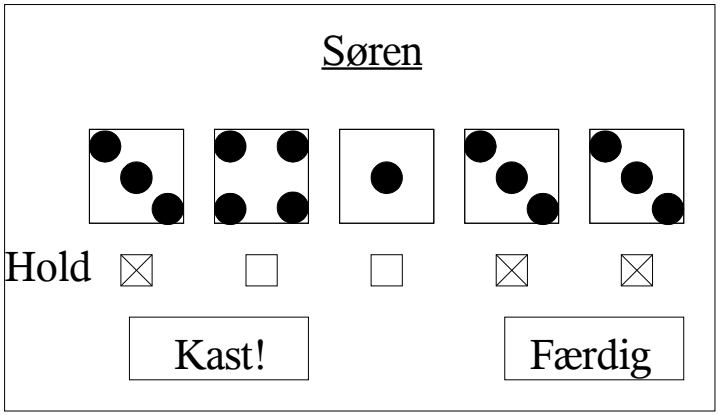






Skærmbilleder





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

Indkapsling

- Veldesignede moduler indkapsler (eng.: encapsulate) informationerne og skjuler deres repræsentation for andre moduler ("klienter"), der bruger modulet
- "Need to know"-princip
  - Hvis et modul ikke har behov for at vide det, får det det ikke at vide
- Klienter skal vide **hvad** et modul gør, ikke **hvordan** det gør det

- Princip:
  - Adskil et moduls (et objekts) implementation fra grænse-fladen (interfacet) til modulet.

Indkapsling med pakker

Adgang til variabler og metoder i et objekt:

Adgang	**public**	**protected**	**(ingenting)**	**private**
i samme klasse	ja	ja	ja	ja
klasse i samme pakke	ja	ja	ja	nej
nedarving i en anden pakke	ja	ja	nej	nej
ej nedarving og 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.

- Et modul, der består af flere klasser, kan lægges i sin egen pakke
  - metoder/variabler, der er interne for modulet erklæres med pakke-synlighed (ingenting)
  - modulets klasser indkapsles derved, så at klasserne kan tilgå hinandens metoder/variabler, mens de ikke er synlige udefra

Designfase

Hvordan skal programmet fungere

Redskaber til objektorienteret design

- Ord til klasser
- Kollaborationsdiagrammer
- Klassediagrammer

Vi bruger nu 10 minutter til hvert punkt!

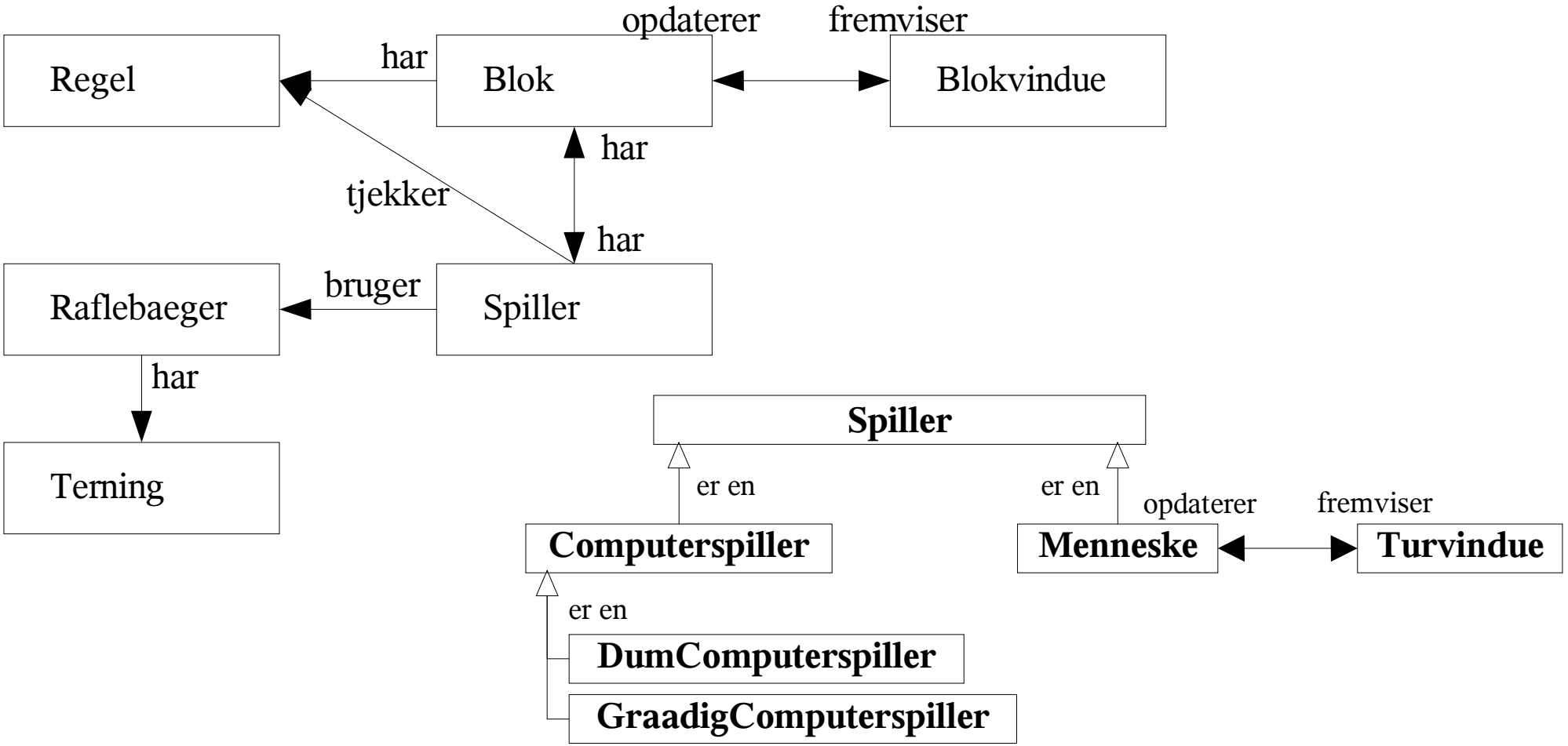
Fremlæggelse næste gang!

Ord til klasser

- Tommelfingerregler:
  - Navneord (substantiver) i ental bliver ofte til klasser
  - Klassenavne skal altid være i ental
  - Udsagnsord (verber) bliver ofte til metoder

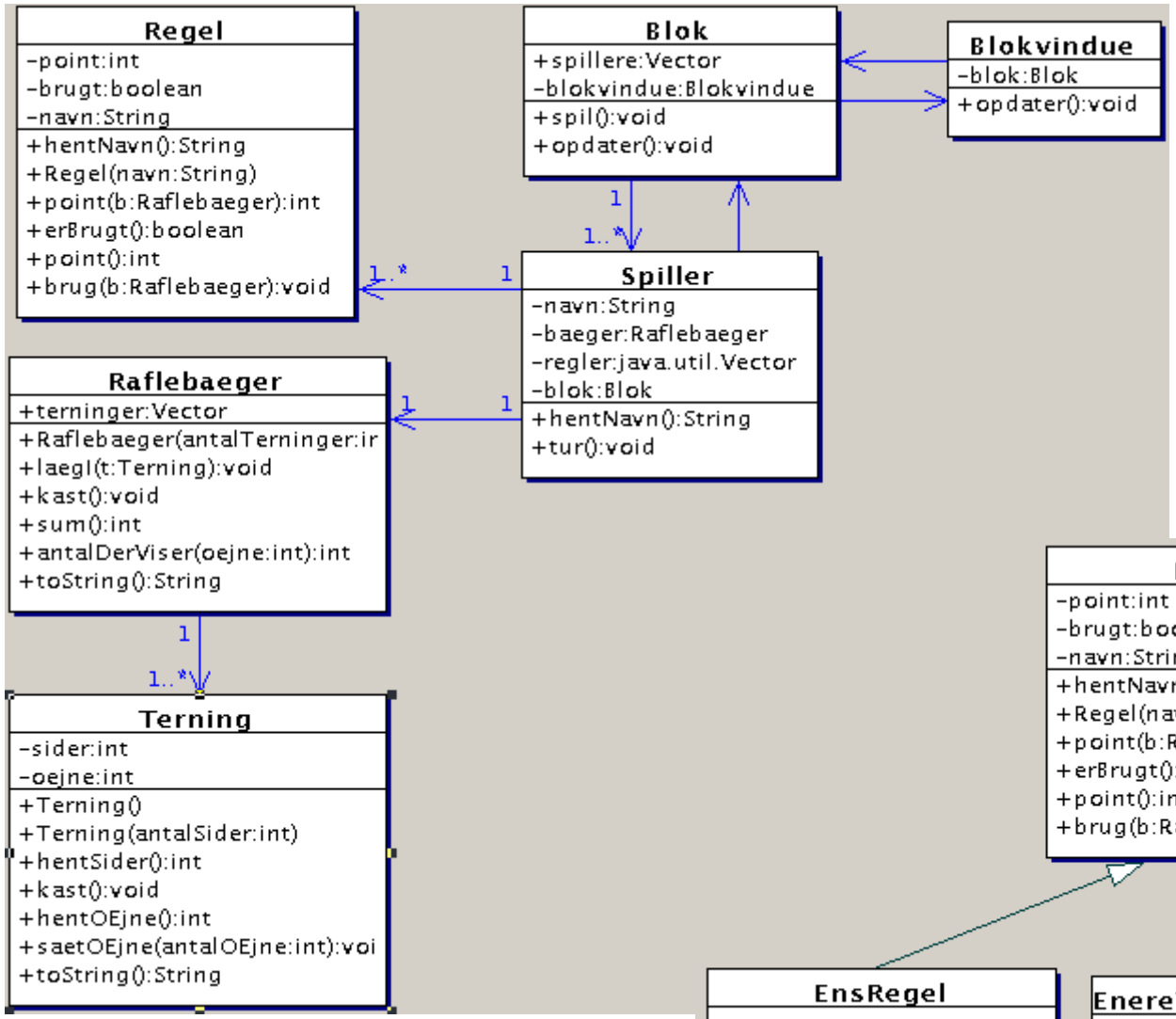
Kollaborationsdiagrammer

- Husk ansvarsområder
- Forskelligt antal => forskellige klasser
- Tegn (har)relationer på



Klassediagrammer

- To-vejs-redigering med et UML-udviklingsværktøj
  - TogetherJ
  - JDeveloper
  - PoseidonUML



Afprøvning (test)

- Vigtigt hvis man ønsker programmer af høj kvalitet
- Formål: At finde (logiske) fejl
  - Psykologisk modstrid
    - Andre end programmøren bør (også) stå for afprøvningen!
- Jo tidligere fejl findes, desto lettere (og dermed billigere) er det at finde dem og rette dem.
- Afprøvning bør foregå på alle stadier af udviklingsprocessen.
- Regression
  - Tidligere test bør gentages hver gang der er foretaget ændringer!

Vigtigste måder at afprøve på

- Modultest (eng.: Unit testing)
  - Hver enhed (klasse eller gruppe af samhørende klasser) afprøves uafhængigt før enhederne integreres i hele systemet.
- Integrations- og systemtest:
  - Afprøve delene og hele systemet samlet.
- Accepttest/slutbrugertest:
  - Sikrer at systemet opfører sig som forventet af brugeren.

Modultest

- Afprøve de enkelte moduler
  - Formål: Verificere at modulet er i overensstemmelse med moduldesignet
  - Modul = klasse eller gruppe af samhørende klasser
- JUnit
  - En ramme for modultest (kan dog også bruges til integrationstest)

JUnit i praksis

- Demonstration

Integrationstest

- Afprøve samspillet mellem de enkelte moduler
  - Formål: Verificere at de er i overensstemmelse med programdesignet
- Fremgangsmåder
  - Inkrementaltest
    - Et modul adderes ad gangen
      - Nedefra og op
        - Først integreres de grundlæggende moduler og deres samspil testes
      - Oppefra og ned
        - Fordel: Man har tidligt en "skal" af systemet kørende
        - Ulempe: Man må skrive 'test-stubbe' der opfører sig som moduler
  - Totaltest ('big bang'-test)
    - Alle moduler samles og hele systemet testes på én gang
      - Nemt at konstatere fejl
      - Sværere at finde grundene til fejlene
    - Kan *ikke* anbefales i større systemer

Afprøvningsstrategier

- Blackbox testing - grænsefladetest
  - Afprøvning af delens grænseflade til resten af systemet.
  - Kontrollerer at parameteroverførsler er som specificeret
- Whitebox testing - test
  - Afprøvning sker i forhold til at man ved hvordan delen er implementeret (den indre logik og kodestruktur).

Blackbox Testing

- Ækvivalensinddeling:
  - Mængden af det mulige input inddeles i ækvivalensklasser.
  - Grænseværdier bør afprøves.
  - Mængden af det mulige input bør indeholde både gyldigt og ugyldige input.
- Positiv afprøvning:
  - Testtilfælde som forventes at gå godt.
- Negativ afprøvning:
  - Testtilfælde som forventes at gå galt.

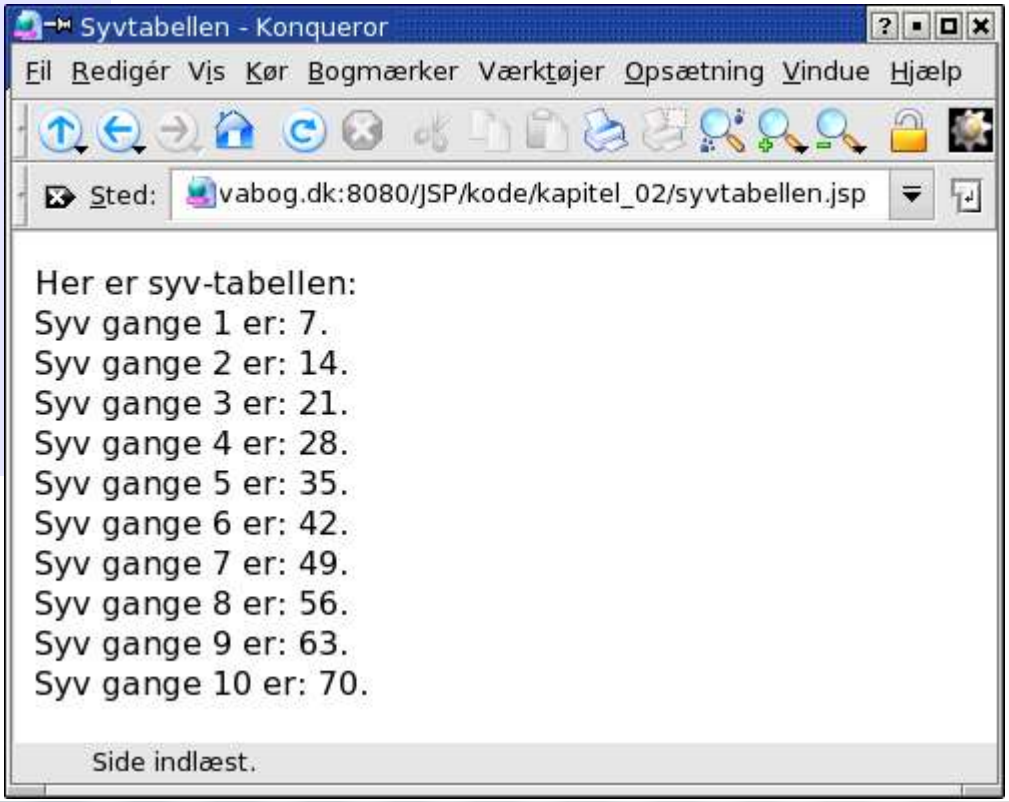
Udvidelser til JUnit

- HttpUnit
  - Til webbaserede systemer
- Cactus
  - Til EJB
  - Til webbaserede systemer
- Borland JDBCFixture
  - Tjek af databaseændringer
- Borland ComparisonFixture
  - Sammenligning af output med en fil

Java Server Pages

- Løkker og kontrolstrukturer skrives i Java
- Udtryk kan også indlejres med <%= %>

```
<html>
<head><title>Syvtabellen</title></head>
<body>
<p>Her er syv-tabellen:<br>

<%
  for (int i=1; i<=10; i++)
  {
%>
   Syv gange <%= i %> er: <%= 7*i %>.<br>
<%
  }
%>
</p>
</body>
</html>
```



request-objektet

```
<html>
<head><title>Data om klienten</title></head>
<body>
<h1>Nogle data om klienten (request-objektet)</h1>
<pre>
URL       - request.getRequestURL(): <%= request.getRequestURL() %>

Metode    - getMethod():     <%= request.getMethod() %>
Protokol  - getProtocol():   <%= request.getProtocol() %>
Værtsnavn - getServerName(): <%= request.getServerName() %>
Port      - getServerPort(): <%= request.getServerPort() %>
URI       - getRequestURI(): <%= request.getRequestURI() %>

Klients IP-adresse - getRemoteAddr(): <%= request.getRemoteAddr() %>
Klients maskinnavn - getRemoteHost(): <%= request.getRemoteHost() %>
Foretrukne sprog   - getLocale():     <%= request.getLocale() %>
Netlæser/browser   header user-agent: <%= request.getHeader("user-agent") %>
</pre>
</body>
</html>
```

Installation af webapplikation

- WAR-fil (Web ARchive)
  - = ZIP-fil med hel applikation
  - Hent WAR-fil
  - Læg i webapps/ og den installeres automatisk
    - Skolens webserver: Åbn http://pingo.cv.ihk.dk:8080/manager/html/ og angiv brugerID stud og adgangskode studx1.
      - Gå ned til "Upload a WAR file to install" og installér
      - MySQL findes på pingo – se i øvelserne hvordan data overføres til Pingo
  - Al opsætning findes i WEB-INF/web.xml

- Hente parametre fra web.xml
  - application - fælles for alle sider
  - config - for en enkelt side

Hente parametre fra web.xml

application-objektet: konfiguration fælles for alle sider.

Lave en indgang af typen context-param i web.xml:

```
<web-app>
  ...
  <context-param>
    <param-name>navnet</param-name>
    <param-value>værdien</param-value>
  </context-param>
  ...
</web-app>
```

Nu vil man i en JSP-side kunne hente værdien af initialiseringsparameteren med:

```
    String værdi = application.getInitParameter("navnet")
```

hvorefter strengen vil have indholdet 'værdien'.

Fra en servlet er application-objektet er tilgængeligt med:

```
    ServletContext application = getServletContext();
```

hvorefter værdien fås som i en JSP-side med:

```
    String værdi = application.getInitParameter("navnet")
```

Hente parametre fra web.xml

config-objektet: konfiguration for den enkelte JSP-side eller servlet.

Lave en indgang af typen context-param i web.xm under en side:

```
<web-app>
  <servlet>
    <servlet-name>En simpel servlet</servlet-name>
    <servlet-class>SimpelServlet</servlet-class>
    <init-param>
      <param-name>navnet</param-name>
      <param-value>værdien</param-value>
    </init-param>
  </servlet>

  <servlet>
    <servlet-name>jspside</servlet-name>
    <jsp-file>/jspside.jsp</jsp-file>
    <init-param>
      <param-name>navnet</param-name>
      <param-value>værdien</param-value>
    </init-param>
  </servlet>

  <servlet-mapping>
    <servlet-name>En simpel servlet</servlet-name>
    <url-pattern>/servlet/SimpelServlet</url-pattern>
  </servlet-mapping>
</web-app>
```

Fra servletten SimpelServlet kunne dette hentes med

```
    String værdi = getInitParameter("navnet");
```

Fra JSP-siden jspside.jsp ville det kunne dette hentes med

```
    config.getInitParameter("navnet")
```

URL'er til websider (web.xml)

```
<web-app>
  <servlet>
    <servlet-name>En simpel servlet</servlet-name>
    <servlet-class>SimpelServlet</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>En simpel servlet</servlet-name>
    <url-pattern>/servlet/SimpelServlet</url-pattern>
  </servlet-mapping>

  <servlet-mapping>
    <servlet-name>En simpel servlet</servlet-name>
    <url-pattern>/simpel/*</url-pattern>
  </servlet-mapping>

  <servlet>
    <servlet-name>jspside</servlet-name>
    <jsp-file>/jspside.jsp</jsp-file>
  </servlet>

  <servlet-mapping>
    <servlet-name>jspside</servlet-name>
    <url-pattern>/jspsider/*</url-pattern>
  </servlet-mapping>

</web-app>
```

Pakning af webapplikation

- Filstruktur i WAR-fil:

```
     0  09-27-03 12:23   WEB-INF/
     0  09-27-03 12:23   WEB-INF/lib/
158892  08-13-01 21:42   WEB-INF/lib/log4j.jar
 56691  07-16-03 18:34   WEB-INF/lib/oscache.jar
218325  09-27-03 12:23   WEB-INF/lib/JSPWiki.jar
 14762  07-16-03 18:44   WEB-INF/jspwiki.properties
  9524  08-13-03 23:04   WEB-INF/jspwiki.tld
  4031  11-06-02 21:22   WEB-INF/web.xml
  2697  08-13-03 22:58   Diff.jsp
  6169  03-08-03 11:49   Edit.jsp
  1784  03-13-03 23:29   Error.jsp
  1572  02-17-03 12:49   PageInfo.jsp
  2212  02-17-03 12:49   PageModified.jsp
  1529  03-29-03 11:15   Preview.jsp
  1802  02-17-03 12:49   Search.jsp
  1508  02-17-03 12:49   Upload.jsp
  2159  02-17-03 12:49   UserPreferences.jsp
  2095  05-23-03 00:37   Wiki.jsp
     0  09-27-03 12:02   images/
   842  02-17-03 12:49   images/attachment_big.png
   178  02-17-03 12:49   images/attachment_small.png
   927  05-23-03 00:01   images/out.png
   396  06-09-02 15:33   images/xml.png
  1182  06-09-02 15:33   images/xmlCoffeeCup.png
```

Servletter

- En servlet er en Java-klasse der bliver brugt i en webserver.
  - Servletten skal arve fra HttpServlet
  - Servletten skal have en doGet(req, resp)-metode
    (eller en doPost()-metode)
  - Response-objektet bruges til at skrive HTML-kode i.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class SimpelServlet extends HttpServlet
{
 public void doGet(HttpServletRequest request,
                   HttpServletResponse response) throws IOException
 {
  response.setContentType("text/html");
  PrintWriter ud = response.getWriter();
   ud.println("<html>");
   ud.println("<head><title>Hej verden</title></head>");
   ud.println("<body>");
   ud.println("<h3>Hej verden!</h3>");
   ud.println("Simpelt eksempel på en servlet");
   ud.println("</body>");
   ud.println("</html>");
 }
}
```

Servletter

- Opsætning af servlet i web.xml

1. Navnet på servletten i <servlet-name>
2. Klassenavnet (incl. pakkenavn) i <servlet-class>
3. Hvilke(n) URL(er) på serveren der skal omdirigeres til servletten i <url-pattern> i en <servlet-mapping>

```
<web-app>
  ...
  <servlet>
    <servlet-name>En simpel servlet</servlet-name>
    <servlet-class>SimpelServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>En simpel servlet</servlet-name>
    <url-pattern>/servlet/SimpelServlet</url-pattern>
  </servlet-mapping>
  ...
</web-app>
```

Nu kan servletten ses på http://maskine.dk/servlet/SimpelServlet

Servletter

- <url-pattern> giver en række muligheder
  - Flere URL'er til det samme:

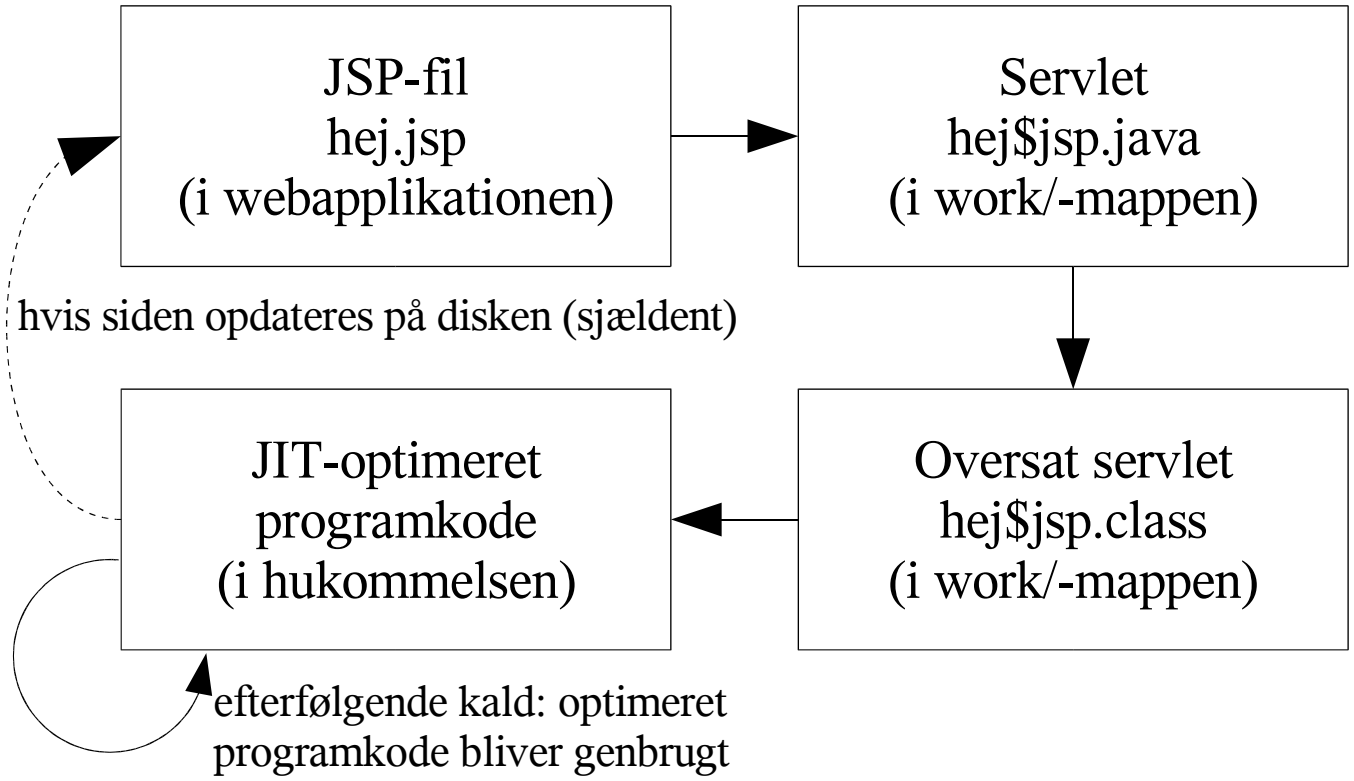
```
<web-app>
  ...
  <servlet>
    <servlet-name>En simpel servlet</servlet-name>
    <servlet-class>SimpelServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>En simpel servlet</servlet-name>
    <url-pattern>/simpel/*</url-pattern>
  </servlet-mapping>
  ...
</web-app>
```

  - automatisk binding af servletter
    - frarådes i drift (sikkerhedsproblemer)

```
  <servlet>
    <servlet-name>invoker</servlet-name>
    <servlet-class>org.apache.catalina.servlets.InvokerServlet</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>invoker</servlet-name>
    <url-pattern>/servlet/*</url-pattern>
  </servlet-mapping>
```

JSP-siders livscyklus

- JSP-sider oversættes internt til servletter
  - Langsomt første gang
  - Hurtig udførelse efterfølgende
  - Fejlmeddelelser ikke altid nemme at forstå

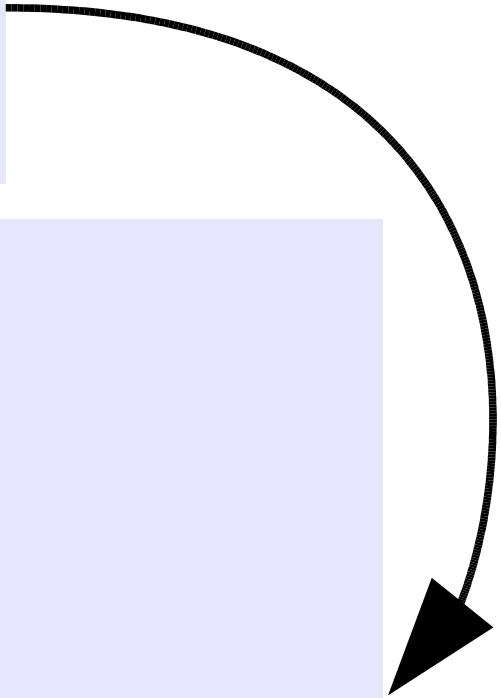


```
<html>
<head><title>Syvtabellen</title></head>
<body>
<p>Her er syv-tabellen:<br>

<%
  for (int i=1; i<=10; i++)
  {
%>
    Syv gange <%= i %> er: <%= 7*i %>.<br>
<%
  }
%>
</p>
</body>
</html>
```

Oversættes internt til

```
package org.apache.jsp.kode.kapitel_02;

import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;

public final class syvtabellen_jsp extends org.apache.jasper.runtime.HttpJspBase {
  public void _jspService( HttpServletRequest request,
                           HttpServletResponse response)
        throws java.io.IOException, ServletException {
    JspFactory _jspxFactory = null;
    PageContext pageContext = null;
    HttpSession session = null;
    ServletContext application = null;
    ServletConfig config = null;
    JspWriter out = null;
    Object page = this;
    JspWriter _jspx_out = null;

    try {
      _jspxFactory = JspFactory.getDefaultFactory();
      response.setContentType("text/html");
      pageContext = _jspxFactory.getPageContext(this, request, response,
          null, true, 8192, true);
      application = pageContext.getServletContext();
      config = pageContext.getServletConfig();
      session = pageContext.getSession();
      out = pageContext.getOut();
      _jspx_out = out;

      out.write("<html>\n");
      out.write("<head><title>Syvtabellen</title></head>\n");
      out.write("<body>\n");
      out.write("<p>Her er syv-tabellen:<br>\n");
```

```
<html>
<head><title>Syvtabellen</title></head>
<body>
<p>Her er syv-tabellen:<br>

<%
 for (int i=1; i<=10; i++)
 {
%>
   Syv gange <%= i %> er: <%= 7*i %>.<br>
<%
 }
%>
</p>
</body>
</html>
```



```
    out.write("<head><title>Syvtabellen</title></head>\n");
    out.write("<body>\n");
    out.write("<p>Her er syv-tabellen:<br>\n");
    out.write("\n");

 for (int i=1; i<=10; i++)
 {
    out.write("\n");
    out.write("\t\tSyv gange ");
    out.print( i );
    out.write(" er: ");
    out.print( 7*i );
    out.write(".<br>\n");
 }

    out.write("\n");
    out.write("</p>\n");
    out.write("</body>\n");
    out.write("</html>");
  } catch (Throwable t) {
    if (!(t instanceof SkipPageException)){
      out = _jspx_out;
      if (out != null && out.getBufferSize() != 0)
        out.clearBuffer();
      if (pageContext != null) pageContext.handlePageException(t);
    }
  } finally {
    if (_jspxFactory != null) _jspxFactory.releasePageContext(pageContext);
  }
 }
}
```

Implicit definerede objekter

- Der findes en række implicit definerede objekter, som man altid har adgang til i en JSP-side:
  - **request - anmodningen fra klienten**
  - **response - svaret til klienten**
  - **out - skrive tekst til klienten**
  - **session - objekt der følger den enkelte bruger**
  - **application - fælles for hele webapplikationen**
    - **logning**
    - **konfigurations-parametre fra web.xml**
    - **kan også gemme attributter ligesom session-objektet**
  - config - den enkelte websides konfiguration
  - page - selve JSP-siden
  - exception - undtagelse opstået under kørsel
  - pageContext - alle objekterne samlet i ét