

Java WebStart

Java WebStart er et verktøy for å distribuere Java-applikasjoner av ulik type.

Forfatter: Else Lervik (else.lervik@hist.no)

Dato: 2006-03-01

Innhold

1	Introduksjon.....	1
2	Hva må utvikleren gjøre?.....	2
3	Konfigurasjon av webtjeneren	3
4	Java Kontrollpanel	3
5	Jnlp-filen.....	3
6	Eksempler.....	4
6.1	<i>Eksempel 1 – applikasjon av enkleste slag.....</i>	<i>5</i>
6.2	<i>Eksempel 2 – klasser som bruker ressurser</i>	<i>5</i>
6.3	<i>Eksempel 3 – tilgang til lokalt filsystem.....</i>	<i>6</i>
6.4	<i>Eksempel 4 – kommunikasjon med tjenerprogram og rmi-register hos tjeneren.....</i>	<i>7</i>
6.5	<i>Eksempel 5 – stort eksempel med pakker og ressurser</i>	<i>8</i>
6.6	<i>Eksempel 6 – kommunikasjon med database</i>	<i>10</i>
7	Litt om digitale signaturer og sertifikater	11
8	Referanser	12

1 Introduksjon

Med Java Webstart installert kan brukere kjøre Java-applikasjoner på en enkel måte, for eksempel ved å klikke på et ikon på skjermen. Nye versjoner lastes ned automatisk fra nettet.

Første gang en bruker skal kjøre et program, må han laste det ned fra en web-side. Når han starter programmet ved en senere anledning, begynner det alltid med å sjekke om ny versjon er publisert. Hvis det er tilfelle, lastes den ned uten at brukeren merker det. Dersom brukeren er frakoplet, oppdager programmet dette og starter umiddelbart med den versjonen som allerede er lastet ned. Oppdatering skjer alltid via web, og det at brukeren er tilkoplet betyr at han er på Internett og at den aktuelle web-tjeneren er oppe.

Brukerne kan redigere mengden av applikasjoner ved å bruke Java Control Panel (Windows: Kontrollpanel/Java).

Om nødvendig sørger WebStart for å laste ned riktig utgave av Java Runtime Environment (JRE).

Programmene kjører i en sandkasse på samme måte som appleter. Det vil si at de ikke har tilgang til lokale filer eller nettverket.

Utviklerne lager programmene på samme måte som ellers. Imidlertid må klassene pakkes i jar-filer, og dette må den som programmerer ta hensyn til i enkelte tilfeller, som vi skal komme tilbake til etter hvert. Ved å signere jar-filene kan en omgå sandkassomodellen og for eksempel skrive til filer på lokal disk.

Utviklerne må lage en webside som brukerne benytter for å få tak i applikasjonene. Denne kan være veldig enkel, men den må inneholde en lenke til en såkalt jnlp-fil som beskriver applikasjonen.

Dette notatet krever at klienten har installert JavaWebStart. Programmet er en del av Java Runtime Environment, og dermed er det vel installert hos de fleste (?). Det er mulig å lage script som sjekker dette og om nødvendig sørger for nedlasting og installasjon, se referanselisten bakerst i notatet.

2 Hva må utvikleren gjøre?

En utvikler som skal distribuere en Java-applikasjon, må kort fortalt gjøre følgende:

1. Hvis bildefiler o.l. benyttes i Java-koden, må disse tas inn via `getResource()`, se eksempel i kapittel 6.2.
2. Samle class-filer og ressurser (f.eks. bildefiler) i en katalog, eventuelt med underkataloger. Lag en jar-fil slik:

```
>jar cf eksempel.jar *
```

Her vil alle filer og underkataloger med filer bli med i jar-filen. `c` står for create, dvs. lag ny pakke, mens `f` betyr at vi skal lage en ny fil i motsetning til å liste pakken ut i konsollvinduet (!).

Merk at det ikke er nødvendig å oppgi klassen der `main()` ligger i manifestet i jar-filen. Dette settes opp i jnlp-filen i stedet. Se pkt. 3.

Husk at Java-programmene som du kjører lokalt på maskinen din, ofte bruker mange klasser som de har tilgang til via `CLASSPATH`. Disse klassene må du sørge for å få med i jar-filene som skal distribueres via WebStart.

3. Lag jnlp-fil der blant annet alle jar-filene ramses opp. Husk å ta med databasedriverer, etc. Navnet på klassen som inneholder `main()` skal ligge i jnlp-filen. Denne filen gjennomgås nærmere i kapittel 5.
4. Lag webside (html-fil) med lenke til jnlp-filen. Brukeren laster ned applikasjonen via denne html-siden.
5. Hvis applikasjonen kommuniserer med lokalt filsystem eller nettverket (for eksempel RMI-tjener eller databasetjener), må du

- a. signere alle jar-filene, eksempel

```
>jarsigner eksempel.jar navn
```

der `navn` er aliaset for nøkkelparet (se kapittel 7)

- b. legge inn følgende element i jnlp-filen

```
<security>
  <all-permissions/>
</security>
```

3 Konfigurasjon av webtjeneren

Webtjeneren må konfigureres til å tolke jnlp-filer som MIME-typen *application/x-java-jnlp-file*. Tomcat-tjeneren er satt opp med dette som standard, se filen *tomcat/conf/web.xml* (fra linje 535 i min installasjon):

```
<mime-mapping>
  <extension>jnlp</extension>
  <mime-type>application/x-java-jnlp-file</mime-type>
</mime-mapping>
```

4 Java Kontrollpanel

Dette er brukerens verktøy for å holde oversikt over Java-applikasjonene. På en PC startes det med menyvalget Kontrollpanel/Java:



klikk her, og velg deretter View Applications for å se hvilke applikasjoner som er lastet ned

I de menyene som så kommer opp, kan du blant annet installere snarvei på skrivebordet.

NB! Under utvikling kan det være greit å se Java-konsollet. Velg fanen Advanced, og deretter Java console. Trykk på knappen "Show console".

5 Jnlp-filen

For å få tilgang til applikasjonen første gang må brukeren laste ned en webside. Denne web-siden må inneholde en lenke til en jnlp-fil som inneholder informasjon om applikasjonen. Lenken kan for eksempel se slik ut:

```
<a href="eksempel1.jnlp">Last ned programmet</a>
```

Her følger et eksempel på jnlp-fil av enkleste slag:

```

<?xml version="1.0" encoding="utf-8"?>
<!-- JNLP File for Eksempel 1, arealberegning -->
<jnlp codebase="http://localhost:8080/webstart/eksempel1/"
      href="eksempel1.jnlp">
  <information>
    <title>Arealberegning</title>
    <vendor>http://www2.tisip.no/boker/java/</vendor>
    <offline-allowed/>
  </information>
  <resources>
    <jar href="kode/eksempel1.jar"/>
    <j2se version="1.5+"
          href="http://java.sun.com/products/autodl/j2se"/>
  </resources>
  <application-desc main-class="Arealberegning2"/>
</jnlp>

```

Figuren nedenfor beskriver de ulike jnlp-elementene. Det fins en god del flere muligheter, se syntaksbeskrivelse i referanselisten bakerst i dette notatet.

<pre> <?xml version="1.0" encoding="utf-8"?> <!-- JNLP File for Eksempel 1, arealberegning --> <jnlp codebase="... URL til katalog ..." href="eksempel1.jnlp"> <information> <title>Arealberegning</title> <vendor>http://www2.tisip.no/boker/java/</vendor> <offline-allowed/> </information> <resources> <jar href="kode/eksempel1.jar"/> <j2se version="1.5+" href="http://java.sun.com/products/autodl/j2se"/> </resources> <application-desc main-class="Arealberegning2"/> </jnlp> </pre>		<p>href-attributtet skal gi navnet på jnlp-filen. Dette skal være relativt til stien oppgitt i codebase-attributtet.</p>
	<p>Tittel og leverandør. Obligatorisk.</p>	
<p><offline-allowed/> sier at applikasjonen kan kjøre uten at kontroll av oppdateringer må gjøres</p>		
<p>jar-filen som inneholder applikasjonen. Relativt til codebase.</p>		
	<p>Versjonskrav til JRE. href angir hvorfra den kan lastes ned.</p>	
<p>Den klassen i jar-filen som inneholder main().</p>		

6 Eksempler

De fleste eksemplene er hentet fra boka "Programmering i Java", 3. utgave (Else Lervik og Vegard B. Havdal, Stiftelsen TISIP og Gyldendal Akademisk, 2004).

Under utvikling har jeg hatt dårlig erfaringer med Opera 8.5 på grunn av utstrakt bruk av caching i denne nettleseren. Internet Explorer 6.0 fungerer fint.

Eksemplene (inkl. jnlp-filer, etc.) kan lastes ned fra <http://www2.tisip.no/boker/java/div/webstart.zip>.

Eksempelfilen pakkes opp og legges i katalogen webapps/ROOT/ under <tomcat-home>.

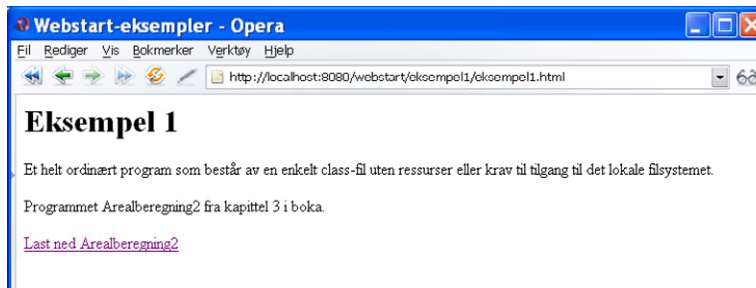
Sjekk at Tomcat er riktig konfigurert (se kapittel 3) og start Tomcat-tjeneren.

Da skal eksemplene kunne hentes opp på adressen `http://localhost:8080/webstart/eksempelN/eksempelN.html` der N er 1, 2, 3, 4, 5 eller 6.

For å kjøre eksempel 3, 4 og 6 må du signere jar-filene (se kapittel 7).

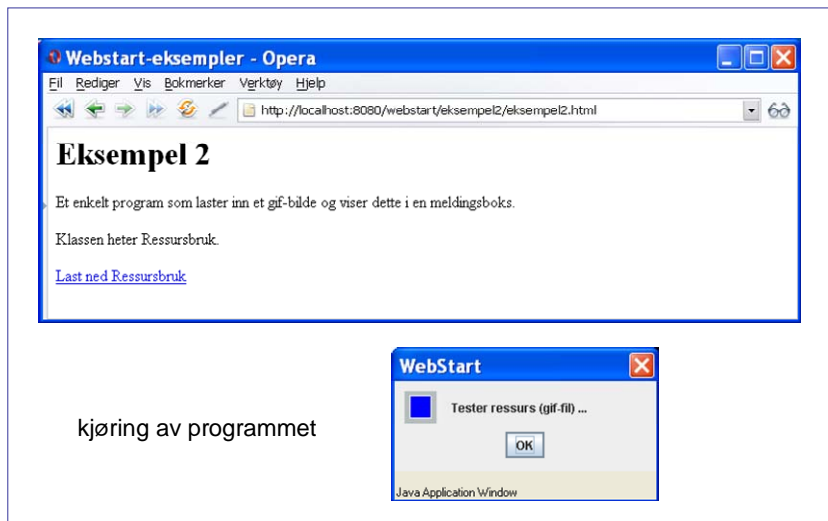
Du bør kjøre eksemplene med åpent Java-konsoll på klientsiden. Se siste del av kapittel 4.

6.1 Eksempel 1 – applikasjon av enkleste slag



Dette er et helt enkelt eksempel der Java-programmet består av en enkelt klasse. `jnlp`-filen i dette eksemplet er den samme som er vist i kapittel 5.

6.2 Eksempel 2 – klasser som bruker ressurser



Den blå firkanten er en gif-fil. Vi forutsetter nå at gif-filen og class-filen ligger på samme katalog. Programmet som kjører lokalt ser slik ut:

```
import javax.swing.*;
class Ressursbruk {
    public static void main(String[] args) {
        Icon bilde = new ImageIcon("blaa.gif"); // gif-filen i samme katalog som class-filen
        JOptionPane.showMessageDialog(null, "Tester ressurs (gif-fil) ...", "WebStart", 0, bilde);
    }
}
```

Dette vil ikke fungere dersom gif-filen ligger i en jar-fil. Å oppgi filnavnet slik vi har gjort i koden foran, betyr at vi refererer til det lokale filsystemet, og det gjør vi altså ikke.

I stedet må vi sende en melding til klasselasteren (class loader) om å laste inn gif-filen. Enhver JVM har en innebygget klasselaster. Klasselasteren bruker vanligvis filsystemet til å finne class-filer med de oppgitte navnene, og som oftest trenger vi ikke tenke på den. Klasselasteren klarer også å finne class-filer som ligger i en jar-fil. Men den klarer ikke å finne andre typer filer som ligger der. Det må vi be om spesielt ved å sende meldingen `getResource()` til klasselasteren. Det vil si at vi må ha en referanse til klasselasteren. Det får vi ved å bruke `getClassLoader()` på et objekt av klassen.

Koden over må altså skrives om, slik:

```
import javax.swing.*;
class Ressursbruk {
    public static void main(String[] args) {
        Ressursbruk objekt = new Ressursbruk(); // trenger et objekt for å finne klasselasteren
        ClassLoader cl = objekt.getClass().getClassLoader(); // henter referanse til klasselasteren
        Icon bilde = new ImageIcon(cl.getResource("blaa.gif")); // bruker klasselasteren til å hente inn bildet
        JOptionPane.showMessageDialog(null, "Tester ressurs (gif-fil) ...", "WebStart", 0, bilde);
    }
}
```

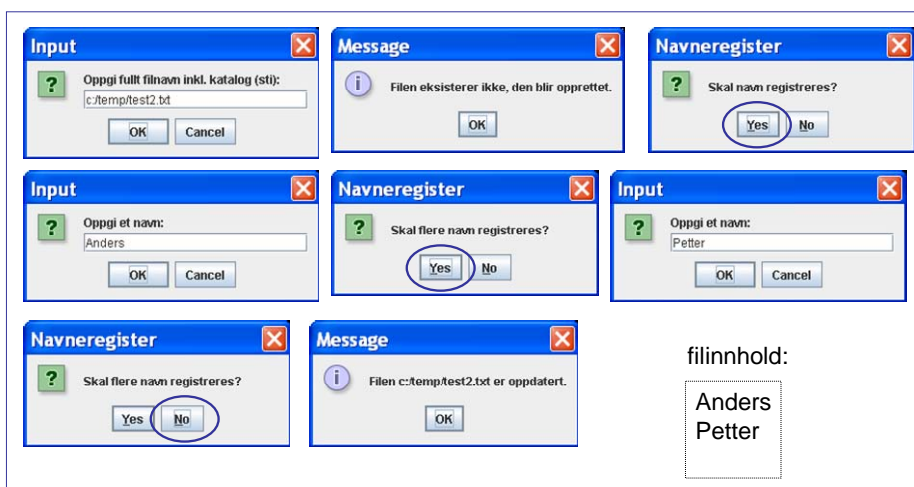
I følge dokumentasjonen kan det eksistere implementasjoner av JVM der `getClassLoader()` returnerer null. Da skal det gå an å bruke klassemetoden `getSystemClassLoader()`, slik:

```
ClassLoader cl = ClassLoader.getSystemClassLoader();
```

Denne måten fungerer ikke i eksemplet foran, det vil si når `getClassLoader()` ikke returnerer null. Med unntak av stier, filnavn og tekster er `inlp`-filen som på side 4.

6.3 Eksempel 3 – tilgang til lokalt filsystem

Vi skal nå se på et program som opererer på brukerens filsystem. Det sier seg selv at brukeren bør være *ekstremt forsiktig* med å laste ned denne typen program. Programmet vi skal se på er omtrent det samme som programliste 12.1 i læreboka. Vi har endret litt på koden slik at filnavnet leses inn og skrives ut helt til slutt. Her er et eksempel på kjøring:



Programmer som distribueres via Java WebStart, kjører i utgangspunktet i en *sandkasse*. Det betyr at programmene ikke har tilgang til lokalt filsystem eller nettverket.

Den som distribuerer programmene kan tilby mottakeren å laste ned programmer som trenger gjennom kantene på sandkassen, ved å signere jar-filen.

Den eneste forskjellen fra de tidligere jnlp-filene, er at vi må legge inn et krav om at det ikke skal være sikkerhetsrestriksjoner på applikasjonen:

```
<security>
  <all-permissions/>
</security>
```

Dette legges inn etter information-elementet.

Jar-filen lager vi på samme måte som tidligere:

```
>jar cf eksempel3.jar *
```

Vi må videre signere filen (merk at dette også gjelder eksempel-filen du har lastet ned, se kapittel 7):

```
>jarsigner eksempel3.jar navn
```

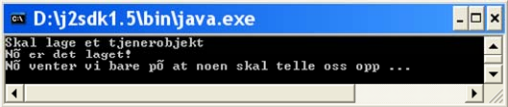
Her er *navn* det aliaset som er definert for den private nøkkelen.

6.4 Eksempel 4 – kommunikasjon med tjenerprogram og rmi-register hos tjeneren

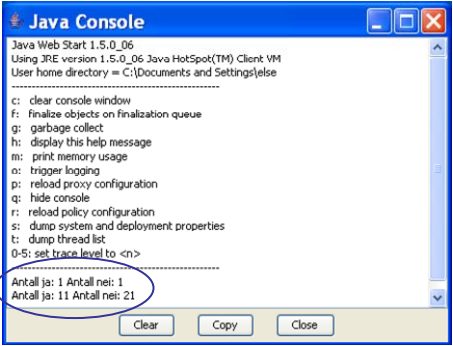
Vi skal nå se på et program som kommuniserer med tjeneren under kjøringen. Vi velger eksemplet i kapittel 19.2 i læreboka.

jar-filen med klientprogramvaren må være signert
jnlp-filen må inneholde security-element

Tjeneren må startes:
start rmiregistry
start java TellerTjener



Klienten starter nedlasting og bekrefter sertifikatet.
Utskriften kommer i Java-konsollet:



Det er kun klientdelen som skal distribueres, det vil si klassene fra følgende java-filer: JaNeiTeller.java og TellerKlient.java. Det enkleste er å legge disse i en egen katalog, kompilere og lage jar-filen på samme måte som tidligere, slik:

```
>jar cf eksempel4.jar *
```

Merk at vi fortsatt kan bruke <offline-allowed/> i jnlp-filen. Programmet krever at vi er på Internett, men det krever ikke direkte tilgang til web-siden (egentlig jnlp-filen) det ble lastet ned fra.

Rmi-kommunikasjonen krever at vi beveger oss utenfor den sandkassen som klientprogrammet normalt kjører i. Vi må derfor signere jar-filen og sette inn security-elementet i jnlp-filen.

```
>jarsigner eksempel4.jar navn
```

Dette gjelder også eksempelfilen du har lastet ned, se kapittel 7.

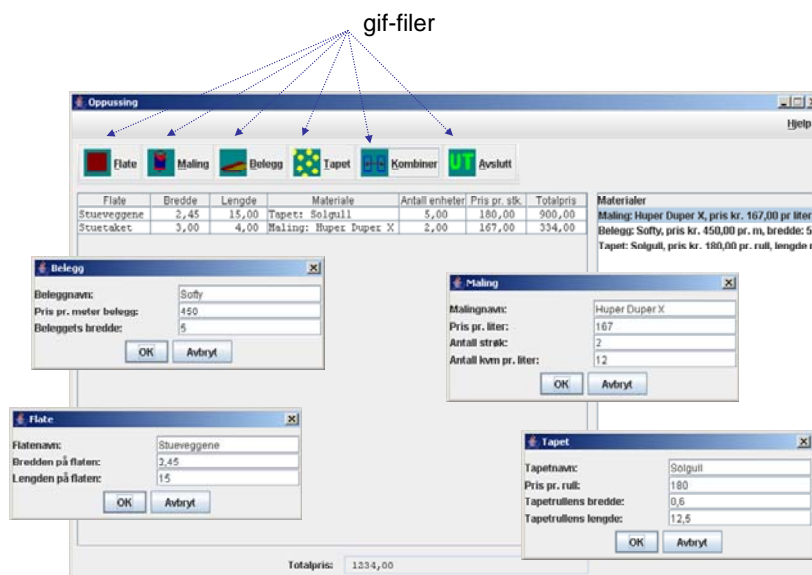
Security-elementet plasseres etter information-elementet, og det ser slik ut:

```
<security>
  <all-permissions/>
</security>
```

6.5 Eksempel 5 – stort eksempel med pakker og ressurser

Her skal vi distribuere eksemplet i kapittel 15.8. Noen av filene i eksemplet er organisert i en pakke, og i tillegg benyttes pakken mittBibliotek. Ikonene i brukergrensesnittet hentes fra gif-filer som ligger i en egen katalog. Alt må ligge i jar-filen. Det enkleste er å kopiere det man trenger av pakken mittBibliotek og legge det samme sted som resten.

Vi legger jar-filen samme sted som java-filene som ikke tilhører navngitte pakker. Der har vi altså underkatalogene ikoner, mittBibliotek og Oppussingsprosjekt.



På grunn av gif-filene må vi gjøre et par små endringer i kildekoden.

Vi har katalogstrukturen i en jar-fil, og der skal / brukes som skille. I filen Konstanter.java refererer vi til java.io.File.separator, som gir verdien "/" i et Windows-miljø. Vi gjør derfor følgende endring:

Linje 9 (10), Konstanter.java:

Det står

```
public static final String ikonkatalog = "ikoner" + java.io.File.separator;
```

Det forandres til:

```
public static final String ikonkatalog = "ikoner/";
```

Enum-klassen *KnappeInfo* er også forandret og tilrettelagt for jar-fil:

```
enum KnappeInfo {
    Flate("Registrer ny flate (golv eller vegg)", 'F', "NyFlate.gif"),
    Maling("Registrer ny malingstype", 'M', "NyMaling.gif"),
    Belegg("Registrer ny type belegg", 'B', "NyttBelegg.gif"),
    Tapet("Registrer ny type tapet", 'T', "NyTapet.gif"),
    Kombiner("Kombiner flate og materiale", 'K', "Kombiner.gif"),
    Avslutt("Avslutt programmet", 'A', "Avslutt.gif");

    private String beskrivelse; // tooltip
    private java.net.URL ikonfil;
    private char mnemonic;

    private KnappeInfo(String startBesk, char startMnem, String startIkonfil) {
        beskrivelse = startBesk;
        mnemonic = startMnem;
        ClassLoader cl = this.getClass().getClassLoader();
        ikonfil = cl.getResource(Konstanter.ikonkatalog + startIkonfil);
    }

    public String finnBeskrivelse() {
        return beskrivelse;
    }

    public java.net.URL finnIkonfil() {
        return ikonfil;
    }

    public char finnMnemonic() {
        return mnemonic;
    }
}
```

Dermed blir det også en liten forandring der ikonet legges ut i vinduet:

Linje 190 (191), filen OppussingKap15GUI.java:

Det står:

```
Icon ikon = new ImageIcon(Konstanter.ikonkatalog + info.finnIkonfil());
```

Det forandres til:

```
Icon ikon = new ImageIcon(info.finnIkonfil()); // argument av typen URL
```

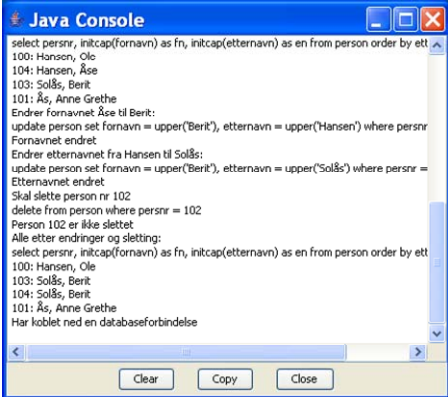
6.6 Eksempel 6 – kommunikasjon med database

Vi skal kjøre eksemplet i kapittel 20.2 via WebStart. Eksemplet kan brukes som det er, men mittBibliotek/Person må legges inn i jar-filen. Som i forrige eksempel er det enklest å legge katalogen mittBibliotek med nødvendig innhold, der resten av koden ligger.

Kildekoden (navn på driver og database) må tilpasses
det databasesystemet du skal bruke.
Kjør eventuelt programmet LagDatabase for å opprette databasen.
Databasen er en liten person-tabell med tre linjer.

jar-filen med klientprogramvaren må være signert
jar-filen med databasedriveren må være signert
jnlp-filen må inneholde security-element og
referanser til begge jar-filene

Klienten starter nedlasting
og bekrefter sertifikatet.
Utskriften kommer i
Java-konsollet:



Merk at vi fortsatt kan bruke `<offline-allowed/>` i jnlp-filen. Programmet krever at vi er på Internett, men det krever ikke direkte tilgang til web-siden (egentlig jnlp-filen) det ble lastet ned fra.

Databasekommunikasjon krever signerte jar-filer. Vi setter opp security-elementet i jnlp-filen:

```
<security>  
  <all-permissions/>  
</security>
```

Så lager vi jar-filen på vanlig måte:

```
>jar cf eksempel6.jar *
```

Databasedriveren, som vanligvis er en jar-fil, må settes opp som egen ressurs i jnlp-filen, det vil si at resources-elementet ser slik ut (forutsatt at databasedriveren heter ojdbc14.jar og ligger i katalogen som er gitt i codebase-attributtet):

```
<resources>  
  <jar href="kode/eksempel6.jar"/>  
  <jar href="ojdbc14.jar"/>  
  <j2se version="1.5+"  
    href="http://java.sun.com/products/autodl/j2se"/>  
</resources>
```

Alle jar-filene må signeres, også ojdbc14.jar:

```
>jarsigner eksempel6.jar navn  
>jarsigner ojdbc14.jar navn
```

Dette gjelder også eksempelfilene du har lastet ned, se kapittel 7.

7 Litt om digitale signaturer og sertifikater

Distributørens *digitale signatur* skal sikre at det er han som har sendt ut jar-filen. Det er greit, men hvordan skal mottakeren gjenkjenne den digitale signaturen? Hvordan kan mottakeren være sikker på at programmet virkelig kommer fra den distributøren han stoler på? Det er her sertifikatene, offentlige og private nøkler, kommer inn i bildet.

Sertifikatutstederen er en tredjepart som begge de involverte partene stoler på. Det er denne instansen som utsteder private og offentlige nøkler, alltid i par med en av hver type. Nøkkelparene er laget slik at den private brukes til kryptering og den offentlige til dekryptering av de samme dataene. Private nøkler skal oppbevares på en betryggende måte hos eieren, mens offentlige nøkler kan distribueres fritt. Sertifikatet er mekanismen som bekrefter at en offentlig nøkkel tilhører en bestemt person.

Hos distributøren benyttes en hash-algoritme til å beregne en kode knyttet til en fil. Det er ikke mulig å gjenskape filen ut fra koden, og det er også slik at hver enkelt fil gir en entydig kode. Denne koden krypteres ved hjelp av den private nøkkelen. Resultatet er den digitale signaturen.

Mottakeren (i vårt tilfelle er det WebStart) beregner også hash-verdien til filen. Deretter dekrypteres signaturen med den offentlige nøkkelen. Dersom resultatet ikke stemmer med den beregnede hash-verdien, er det noe feil et sted. Da bør filen avvises. Den offentlige nøkkelen kan hentes fra sertifikatutstederen, eller den kan følge med filen.

Ulike filer distribuert av den samme distributøren har forskjellige digitale signaturer. I en jar-fil med tre filer vil det finnes tre digitale signaturer.

Å være sin egen sertifikatutsteder

Det fins lister over autoriserte sertifikattilbydere, se for eksempel Java Kontrollpanel/Security/Certificates ... eller oversikter i den nettleseren du bruker (Internet Explorer: Verktøy/Alternativer for Internett/ Innhold/ Utstedere, Opera: Verktøy/ Innstillinger/ Avansert/ Sikkerhet/ Ordne sertifikater/ Autoriteter).

For testformål kan du like gjerne utstede dine egne sertifikater. Bruk verktøyene keytool og jarsigner som følger med Java-kompilatoren og som du har tilgjengelig direkte fra kommandolinjen.

Du bruker keytool til å opprette databasen der sertifikater og nøkler lagres, mens jarsigner benyttes til å hefte digitale signaturer til jar-filer. Disse verktøyene er profesjonelle nok, men her ser vi bare på den aller enkleste bruken. (Dokumentasjon av keytool og jarsigner finner du sammen med den øvrige J2SE-dokumentasjonen.)

Følgende kommando oppretter databasen og genererer et nøkkelpar. Den offentlige nøkkelen legges inn i et sertifikat:

```
>keytool -genkey -alias navn
```

Navn er en entydig identifikasjon av dette nøkkelparet. Egentlig trenger du ikke mer enn ett nøkkelpar. Databasen ligger i filen .keystore som på min PC havnet i katalogen C:\Documents and Settings\else\.

Du vil bli spurt om en del sertifiseringsinformasjon som navn og institusjon. Du må også oppgi passord som skal beskytte databasen og den private nøkkelen. Dette kan være det samme passordet.

Keytool kan også brukes til å administrere sertifikater utstedt av andre.

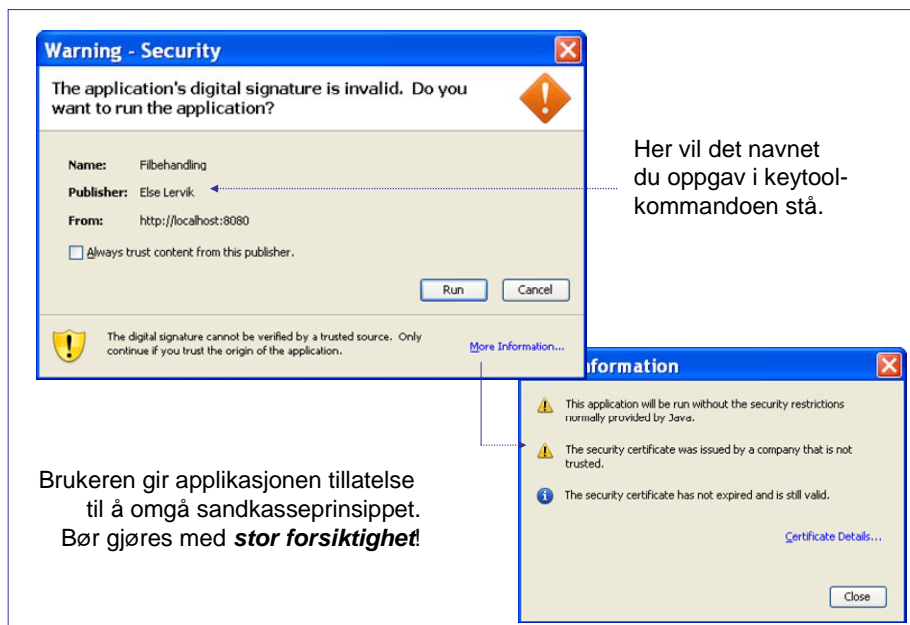
Å signere en jar-fil

Nå som vi har en privat nøkkel, kan vi signere jar-filen:

```
>jarsigner eksempel.jar navn
```

Her er *navn* navnet på nøkkelparet, slik vi oppgav det i keytool-kommandoen. Vi blir spurt om passord.

Når brukeren nå forsøker å hente ned applikasjonen til WebStart, får han følgende bilde på skjermen:



8 Referanser

De fleste eksemplene er hentet fra boka "Programmering i Java", 3. utgave (Else Lervik og Vegard B. Havdal, Stiftelsen TISIP og Gyldendal Akademisk, 2004)

Følgende er benyttet under utarbeidelsen av dette notatet:

- Korte og greie introduksjoner:
 - Deploying Java Web Start Applications:
<http://java.sun.com/docs/books/tutorial/deployment/webstart/deploying.html>
 - Java Web Start Overview White Paper May 2005:
http://java.sun.com/developer/technicalArticles/WebServices/JWS_2/JWS_White_Paper.pdf
- Sun sine sider: <http://java.sun.com/j2se/1.5.0/docs/guide/javaws/>

- Uoffisiell FAQ: <http://webstartfaq.com/>
- Script for å installere WebStart hos brukere som ikke har dette installert:
<http://java.sun.com/j2se/1.5.0/docs/guide/javaws/developersguide/launch.html#intro>
- JNLP File Syntax:
<http://java.sun.com/j2se/1.5.0/docs/guide/javaws/developersguide/syntax.html>
- JNLP API gjør det mulig å programmere bl.a. import og eksport av filer under kontroll av brukeren på tilsvarende måte som i HTML.
<http://java.sun.com/j2se/1.5.0/docs/guide/javaws/jnlp/index.html>
- Om sertifikater og digitale signaturer
 - X.509 Certificates and Certificate Revocation Lists (CRLs):
<http://java.sun.com/j2se/1.5.0/docs/guide/security/cert3.html>
 - Hva er digitale signaturer og PKI? (Arbeids- og administrasjonsdepartementet 2004):
http://odin.dep.no/fad/norsk/dok/andre_dok/nou/002001-020005/hov003-bn.html
- Dokumentasjon av verktøyene *jar*, *keytool* og *jarsigner*:
<http://java.sun.com/j2se/1.5.0/docs/tooldocs/index.html>