Videregående programmering i Java

Dag 13 - valgfrie emner

Evt: Introduktion til J2EE og EJB

Hvordan definere egne generics

Evt.: Værktøjer til forbedring af kodekvalitet - kodemetrikker og audit

Projektvejledning

Udgår: Introduktion til J2ME, midletter og programmering af mobiltelefoner

Læsning: Evt VP 14 (VP 13 udgår)

Sidste gang - spørgsmål?

Dag 12 - Introspektion og optimering

Introspektion af klasser på køretidspunktet (reflektion)
Evt.: JAR-filer og oprettelse af eksekverbare JAR-filer
Optimering af programmer
Evt.: Optimeringsværktøjer (Borland OptimizeIt)
Projektvejledning

Læsning: VP 9, VP 11

Java 2 Enterprise Edition (J2EE) - til store servere

- J2EE-platformens dele
  - Webserver
  - EJB (Enterprise JavaBeans)
  - CORBA - fjernkald af objekter (ikke kun Java, sværere end RMI)
  - JTS (Java Transaction Service) - transaktioner (commit, rollback)
  - JMS (Java Message Service) - til MQ-systemer.
    - tjeneste til at sende og modtage synkrone eller asynkrone meddelelser mellem programmer
  - JavaMail, XML-behandling, ...

- Sun varetager 'kun' J2EE-*specifikationen*
  - Mange uafhængige udbydere af J2EE-platforme
    - Borland AppServer
    - Oracle
    - IBM WebbSphere
    - JBoss (med åben kildekode)
    - Sun (referenceimplementation)
    - ... (>10 andre)

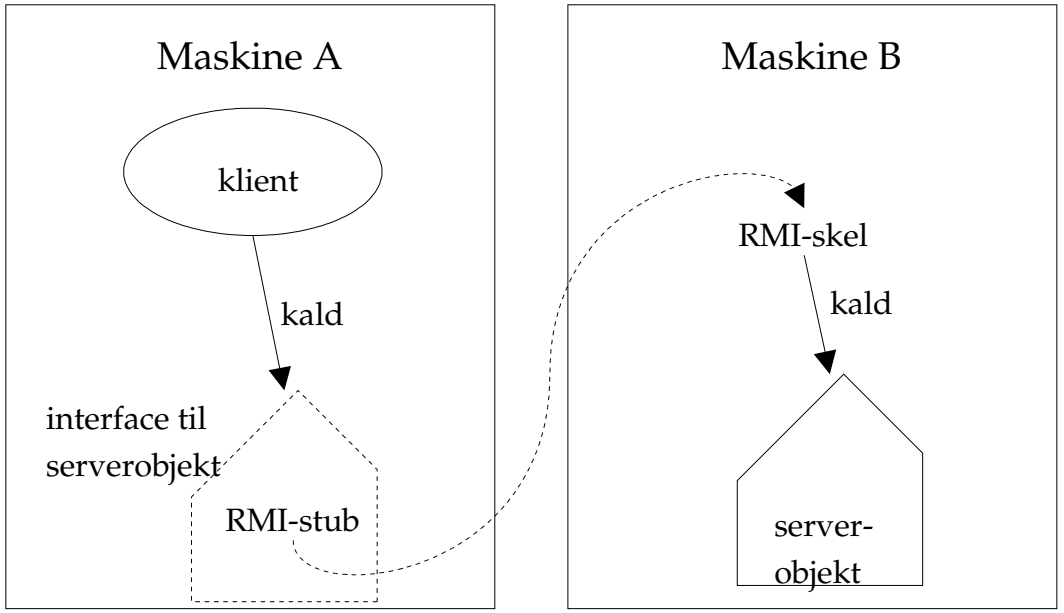
J2EE != udviklingsplatform
Mange udbydere har også egen, 'strømlinet' webudviklingsmodel

Enterprise JavaBeans (EJB) - objekter i serveren

- J2EE-container varetager EJB-objekter
  - Persistens, transaktioner, sikkerhed
  - Kald kan ske lokalt eller over netværket
    - F.eks. fra JSP-sider!

- EJB koncentrerer sig om programlogik
  - Serveren sørger for al 'benarbejdet'
  - Sessionsbønner repræsenterer et forløb
  - Entitetsbønner repræsenterer data fra en tabel
    - Containerstyret persistens (CMP)
      - Al databasekode bliver genereret af containeren (!)
    - Bønnestyret persistens (BMP)
      - Programmøren skriver selv databasekoden
  - Meddelelses-drevne bønner

Principippet i kald af metoder i objekter over netværket



En EJB-bønne kører en en J2EE-applikationsserver - muligvis adskilt fra klienten

Hver EJB-bønne udgøres af tre stykker kildetekst (to interfaces og en klasse):

1) Et **fjerninterface** (eng.: remote interface), der specificerer hvordan bønnen kan bruges af klienten.
2) Et **hjemmeinterface** (eng.: home interface), der specificerer, hvordan bønnen kan fremfindes, oprettes og nedlægges af klienten.
3) En **implementationsklasse** af bønnen, med den programkode, der beskriver, hvad der skal ske, når de forskellige metoder i fjerninterfacet kaldes.

Kildekoden i en EJB

Implementationen af bønnen

```
package vp.ejb;
import java.rmi.RemoteException;
import javax.ejb.*;

public class VekslerBean implements SessionBean
{
   public void ejbCreate() {}
   public void setSessionContext(SessionContext sc) {}
   public void ejbActivate() {}
   public void ejbPassivate() {}
   public void ejbRemove() {}

   public double euroTilDollar(int euro) throws RemoteException
   {
      return euro/1.1;
   }

   public double dollarTilEuro(double dollar) throws RemoteException
   {
      return dollar*1.1;
   }
}
```

Fjerninterfacet (hvordan objektet kan bruges af klienten):

```
package vp.ejb;
import java.rmi.RemoteException;
import javax.ejb.EJBObject;

public interface Veksler extends EJBObject
{
  public double dollarTilEuro(double dollar)
     throws RemoteException;

  public double euroTilDollar(int euro)
     throws RemoteException;
}
```

Hjemmeinterfacet (fremfinding)

```
package vp.ejb;
import java.rmi.RemoteException;
import javax.ejb.*;

public interface VekslerHome extends EJBHome
{
  public Veksler create() throws RemoteException,
                                 CreateException;
}
```

... plus noget information til EJB-containeren (XML)

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 1.1//EN"
"http://java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd">

<ejb-jar>
  <enterprise-beans>
    <session>
      <description>En veksle-bønne (tilstandsløs sessionsbønne)</description>
        <display-name>Veksler</display-name>
        <ejb-name>Veksler</ejb-name>           <!-- JNDI-navn klient bruger -->
        <remote>vp.ejb.Veksler</remote>        <!-- fjerninterfacets navn   -->
        <home>vp.ejb.VekslerHome</home>        <!-- hjemmeinterfacets navn  -->
        <ejb-class>vp.ejb.VekslerBean</ejb-class><!-- klassenavn på bønnen  -->
        <session-type>Stateless</session-type> <!-- bønnen er tilstandsløs  -->
        <transaction-type>Container</transaction-type>
      </session>

<!-- ... beskrivelse af flere bønner kunne komme her ... -->

    </enterprise-beans>
</ejb-jar>
```

Bruge en EJB

Brug af Veksler-bønnen

```
package vp.ejb;
import javax.naming.*;                          // pakken med
JNDI
import javax.rmi.PortableRemoteObject;
public class BenytVeksler
{
  public static void main(String[] args) throws Exception
  {
    // Angiv data til JNDI til dens kontekst.
    // Disse kunne også i stedet angives i jndi.properties

    Hashtable env = new Hashtable();
    env.put(Context.INITIAL_CONTEXT_FACTORY,
         "com.evermind.server.rmi.RMIInitialContextFactory");
    env.put(Context.SECURITY_PRINCIPAL, "admin");
    env.put(Context.SECURITY_CREDENTIALS, "welcome");
    env.put(Context.PROVIDER_URL,
       "ormi://localhost:23891/current-workspace-app");
    Context ctx = new InitialContext(env);

    VekslerHome hjem = (VekslerHome)ctx.lookup("Veksler");

    Veksler valutaveksler =  hjem.create();

    double beløb = valutaveksler.euroTilDollar(100);
    System.out.println("100 euro er "+beløb+" dollar.");

    beløb = valutaveksler.dollarTilEuro(100);
    System.out.println("100 dollar er "+beløb+" euro.");
  }
}
```

Klienten slår først bønnen op med JNDI og får derved fat i hjemmeinterfacet.
Ved hjælp af dette oprettes (en fjernreference til) bønnen, som derefter anvendes.

Hjemmeinterfacet (fremfinding)

```
package vp.ejb;
import java.rmi.RemoteException;
import javax.ejb.*;

public interface VekslerHome extends EJBHome
{
  public Veksler create() throws RemoteException,
                                 CreateException;
}
```

Fjerninterfacet (hvordan objektet kan bruges af klienten):

```
package vp.ejb;
import java.rmi.RemoteException;
import javax.ejb.EJBObject;

public interface Veksler extends EJBObject
{
  public double dollarTilEuro(double dollar)
     throws RemoteException;

  public double euroTilDollar(int euro)
     throws RemoteException;
}
```

Typer af EJB

- Sessionsbønner
  - repræsenterer et eller andet forløb og indeholder de data der knytter sig til dette forløb
    - Tilstandsløse (eksempel: Veksler-bønnen)
      - Flere klienter kan dele samme instans
    - Tilstandsfulde
      - Hver klient får sin egen instans
- Entitetsbønner
  - repræsenterer data fra en tabel. De er *persistente* (dvs. deres data 'bakkes op' af en database).
    - Containerstyret persistens (CMP)
      - Al databasekode bliver genereret af containeren (!)
    - Bønnestyret persistens (BMP)
      - Programmøren skriver selv databasekoden
- Meddelelses-drevne bønner

Entitetsbønner med CMP

Udvikleren designer databasen...

```
CREATE TABLE kunder (navn varchar(32), kredit float)
INSERT INTO kunder VALUES('Jacob', -1799)
INSERT INTO kunder VALUES('Brian', 0)
```

... og beder derefter udviklingsværktøjet generere en Kunde-EJB

Bruge Kunder-objekter fra klienten

```
   Hashtable env = new Hashtable();
   env.put(Context.INITIAL_CONTEXT_FACTORY,
      "com.evermind.server.rmi.RMIInitialContextFactory");
   env.put(Context.SECURITY_PRINCIPAL, "admin");
   env.put(Context.SECURITY_CREDENTIALS, "welcome");
   env.put(Context.PROVIDER_URL,
      "ormi://localhost:23891/current-workspace-app");
   Context ctx = new InitialContext(env);
   KundeHome kunderHome = (KundeHome)ctx.lookup("Kunde");
   Kunde kunder;

   // Use one of the create() methods below to create a new
instance
   // kunder = kunderHome.create(  );
   // kunder = kunderHome.create( java.lang.String navn );

   // Retrieve all instances using the findAll() method
   // (CMP Entity beans only)
   Collection coll = kunderHome.findAll();
   Iterator iter = coll.iterator();
   while (iter.hasNext())
   {
      kunder = (Kunde)iter.next();
      System.out.println("navn = " + kunder.getNavn());
      System.out.println("kredit = " + kunder.getKredit());
      System.out.println();
   }
```

Hjemmeinterfacet (fremfinding)

```
public interface KunderHome extends EJBHome
{
  Kunder create()
     throws RemoteException, CreateException;

  Kunder findByPrimaryKey(String primaryKey)
     throws RemoteException, FinderException;

  Collection findAll()
     throws RemoteException, FinderException;
}
```

Fjerninterfacet (hvordan objektet kan bruges af klienten):

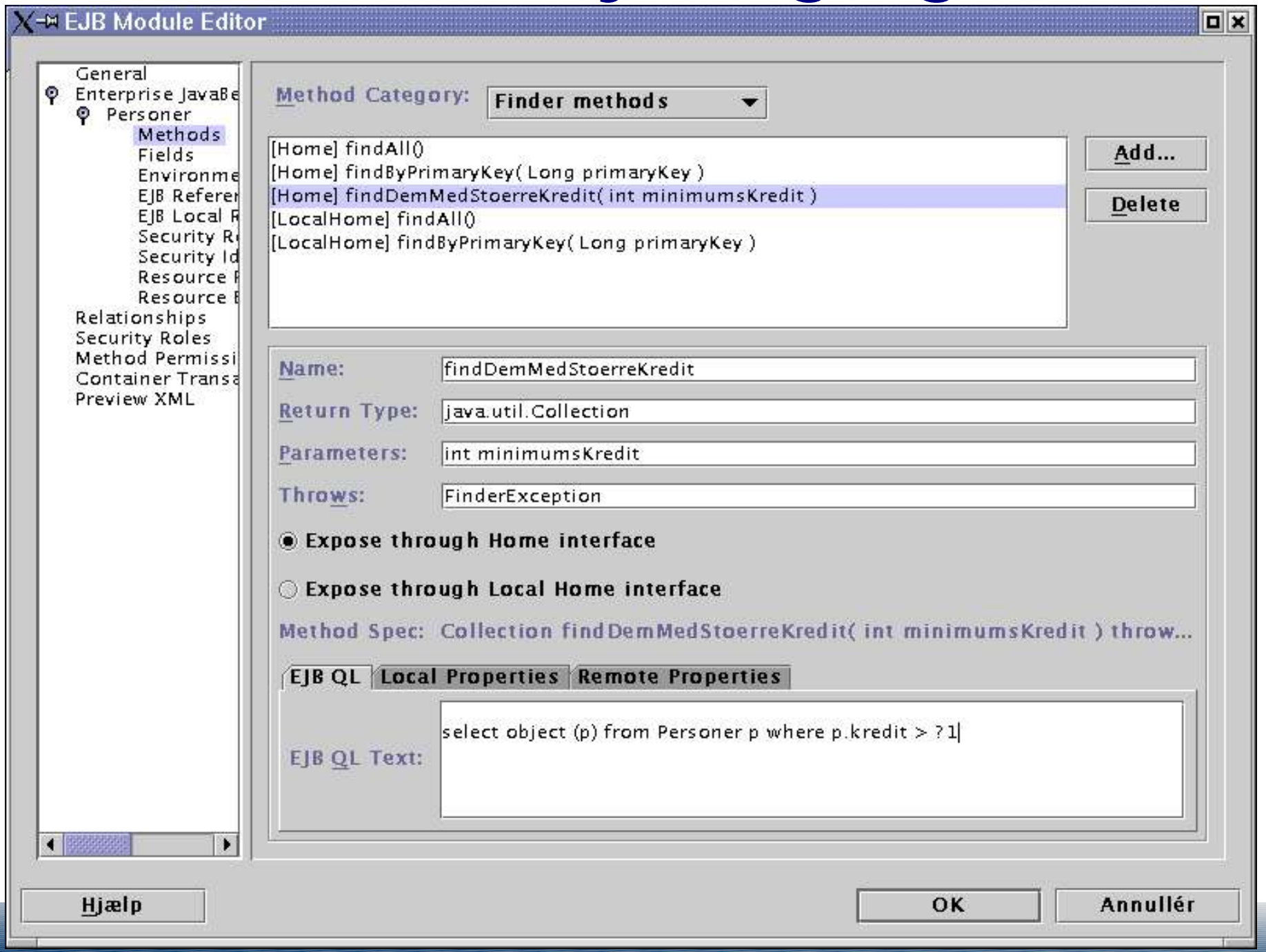
```
public interface Kunder extends EJBObject
{
  String getNavn() throws RemoteException;

  void setNavn(String newNavn) throws RemoteException;

  Float getKredit() throws RemoteException;

  void setKredit(Float newKredit) throws RemoteException
}
```

EJB Query Language



EJB Query Language

```
    <entity>
      <description>Entity Bean ( CMP )</description>
      <display-name>Personer</display-name>
      <ejb-name>Personer</ejb-name>
      <home>mypackage1.PersonerHome</home>
      <remote>mypackage1.Personer</remote>
      <local-home>mypackage1.PersonerLocalHome</local-home>
      <local>mypackage1.PersonerLocal</local>
      <ejb-class>mypackage1.impl.PersonerBean</ejb-class>
      <persistence-type>Container</persistence-type>
      <prim-key-class>java.lang.Long</prim-key-class>
      <reentrant>False</reentrant>
      <cmp-version>2.x</cmp-version>
      <abstract-schema-name>Personer</abstract-schema-name>
      <cmp-field>
        <field-name>personer_pk</field-name>
      </cmp-field>
      <cmp-field>
        <field-name>navn</field-name>
      </cmp-field>
      <cmp-field>
        <field-name>kredit</field-name>
      </cmp-field>
      <primkey-field>personer_pk</primkey-field>
      <query>
        <query-method>
          <method-name>findDemMedStoerreKredit</method-name>
          <method-params>
            <method-param>int</method-param>
          </method-params>
        </query-method>
        <ejb-ql>select object (p) from Personer p where p.kredit > ?1</ejb-ql>
      </query>
    </entity>
  </enterprise-beans>
</ejb-jar>
```

EJB: Transaktioner

- Programmatisk ('i hånden'-) transaktionsstyring
- Deklarativ transaktionsstyring
  - Metoder markeres med en transaktions-attribut
    - Det er containerens opgave at sørge for at oprette, nedlægge, fuldføre og evt. annullere transaktioner i henhold til disse
  - Required - metoden er med i den aktuelle transaktion.
    Der oprettes en ny transaktion af containeren, hvis der ikke er nogen i gang.
  - Supports - metoden er med i den aktuelle transaktion.
    Der oprettes ikke en ny transaktion af containeren, hvis der ikke er nogen i gang.
  - Mandatory - metoden skal være med en transaktion.
    Er der ikke er nogen i gang, kastes undtagelsen TransactionRequiredException.
  - Never - metoden må aldrig være med en transaktion.
    Er der en transaktion i gang, kastes undtagelsen RemoteException af containeren.
  - RequiresNew - der oprettes en ny transaktion.
    Hvis der er en transaktion i gang, suspenderes denne indtil metoden har afsluttet.
  - NotSupported - kan ikke være med en transaktion.
    Hvis der er en transaktion i gang, suspenderes indtil metoden har afsluttet.

Værktøj til forbedring af kodekvalitet – automatisk koderevision (audit)

- Nogle værktøjer (JBuilder, JDeveloper, ...) kan søge programkoden igennem og revidere den (eng.: audit)
  - Består af 30-80 forskellige tjek af koden på forskellig måde
    - Almindelige fejl og uhensigsmæssigheder påpeges
  - Griber ind i programmørens kodningsvaner
    - Nok nødvendigt i projekter med mange programmører
    - Hvilke tjek der skal foretages kan indstilles fra projekt til projekt

Eksempel:

Use Abbreviated Assignment Operator (UAAO)

Use the abbreviated assignment operator in order to write programs more rapidly. Also some compilers run faster when you do so.

Wrong

```
void oper () {
   int i = 0;
   i = i + 20;
   i = 30 * i;
}
```

Right

```
void oper () {
   int i = 0;
   i += 20;
   i *= 30;
}
```

Andre eksempler:

'switch' Statement Should Include a Default Case (SSSIDC)
Assignments to 'for' Loop Variables (AFLV)
Accessing Static Members through Objects (ASMO)
Assignments to Formal Parameters (AFP)

Definere egne generiske klasser

- Parametriserede klasser/generiske typer = erklæring af en klasse med en eller flere klasser som parametre
  - ~= templates fra C++
- Fordele
  - Klarere kode
  - Typesikkerhed

Definere egne generiske klasser

- En generisk Info-klasse

```
package generisk;
import java.awt.*;

/** En klasse til a gemme info om et andet objekt */
public class Info<Obj>
{
  public Obj obj;

  public String txt;

  public Info(Obj o, String t) {
    obj = o;
    txt = t;
  }

  public String toString() {
    return "INFO "+obj.toString()+" "+txt;
  }


  public static void main(String[] args)
  {
    Info<Point> info = new Info<Point>(new Point(), "origo");
    System.out.println(info);
    // udskriver: INFO java.awt.Point[x=0,y=0] origo
  }
}
```

Definere egne generiske klasser

- En generisk serialiserings-klasse
  - Parametriseret ved en klasse, der skal være serialiserbar

```
package generisk;

import java.io.*;

public class Serialisering<Type extends Serializable>
{
  public void gem(Type obj, String filnavn) throws IOException
  {
    FileOutputStream datastrøm = new FileOutputStream(filnavn);
    ObjectOutputStream objektstrøm = new ObjectOutputStream(datastrøm);
    objektstrøm.writeObject(obj);
    objektstrøm.close();
  }

  public Type hent(String filnavn) throws Exception
  {
    FileInputStream datastrøm = new FileInputStream(filnavn);
    ObjectInputStream objektstrøm = new ObjectInputStream(datastrøm);
    Object obj = objektstrøm.readObject();
    objektstrøm.close();
    return (Type) obj;
  }
}
```

```
package generisk;

import java.util.*;

public class HentOgGem
{
  public static void main(String[] arg) throws Exception
  {
    Serialisering<String> serialiseringStr = new Serialisering<String>();
    serialiseringStr.gem("en streng","str.ser");

    //Ulovlig: Serialisering<Socket> serialisering3 = new Serialisering<Socket>();

    Serialisering<ArrayList<String>> serialiseringALS = new Serialisering<ArrayList<String>>()

    ArrayList<String> l;

    try {
      l = serialiseringALS.hent("venner.ser");
      System.out.println("Læst: " + l);
    }
    catch (Exception e) {
      e.printStackTrace();
      l = new ArrayList();
      l.add("Jacob");
      l.add("Brian");
      l.add("Preben");
      System.out.println("Oprettet: " + l);
    }

    l.add("Ven" + l.size());
    serialiseringALS.gem(l, "venner.ser");
    System.out.println("Gemt: " + l);
  }
}
```