Videregående programmering i Java

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

Sidste gang - opsamling

Fra sidst: JNI og kald til maskinkode/C/C++ fra Java

Fra sidst: Internationalisering

De to sidste gange - nogen ønsker?

- Dag 13 - valgfrie emner
  - Evt: Introduktion til J2EE og EJB
  - Evt.: Værktøjer til forbedring af kodekvalitet - kodemetrikker og audit
  - Evt.: Udskrift til printer/sideformattering
  - Projektvejledning

  Udgår: Introduktion til J2ME, midletter og programmering af mobiltelefoner
- Dag 14 – afrunding (og valgfrie emner)
  - Evt.: Nye faciliteter i JDK 1.4, herunder XML-processering, det nye I/O-API, logning og antagelser.
  - Evt.: Nye faciliteter i JDK 1.5, herunder ny syntaks
  - Evt.: AOP - Aspekt-orienteret programmering
  - Om eksamen og eksamensforberedelse
  - Spørgetime
  - Fremlæggelse af projekter

Introspektion af klasser på køretidspunktet (reflektion)

- Introspektion (eng.: Introspection eller reflection):
  - hvordan man under kørslen af programmet kan inspicere et vilkårligt objekt (pakken java.lang.reflect.*)
    - finde dets klasse
    - finde ud af, hvilke metoder og variabler klassen har
    - kalde metoderne og aflæse/sætte variablerne
      - Specielle faciliteter til javabønner (i java.beans.*)
    - oprette nye objekter fra klassen etc. etc.
- Introspektion bruges sjældent, men kan være nyttigt i specielle tilfælde:
  - Udviklingsværktøj
    - debuggerens mulighed for introspektion af variabler
    - GUI-design med (alle mulige) javabønner
  - Serialisering (og dermed RMI)
    - XML-serialisering af javabønner (XMLEncoder) anvender det også
    - Overkommeligt at skrive egen 'serialisering' til at hente/gemme data
- Det er godt at kende til mulighederne for introspektion for at forstå, hvordan andre klasser fungerer

Introspektion

```
import java.lang.reflect.*;
import java.awt.*;

public class UndersoegKlasse
{
    public static void main(String[] args)
    {
       Object o = new Frame();

       // Find klassen
       Class klasse = o.getClass();
       System.out.println("Klassen navn er: "+klasse.getName());

       // Find superklasserne
       Class superklasse = klasse.getSuperclass();
       while (superklasse != null)
       {
          System.out.println("... og den har superklasse: "+superklasse.getName());
          superklasse = superklasse.getSuperclass();
       }
    }
}
```

```
Klassen navn er: java.awt.Frame
... og den har superklasse: java.awt.Container
... og den har superklasse: java.awt.Component
... og den har superklasse: java.lang.Object
```

Introspektion

Vigtigste metoder i Class (der repræsenterer en klasse)

String getName()	giver en streng med klassens navn (og pakkenavn)
Class getSuperclass()	giver Class-objektet, der repræsenterer superklassen
Class[] getInterfaces()	giver et array med de interfaces, klassen implementerer
Field[] getFields()	giver et array med de variabler, der er erklæret public i klassen
Constructor[] getConstructors()	giver et array med de konstruktører, der er erklæret public
Method[] getMethods()	giver et array med de metoder, der er erklæret public i klassen
Class[] getClasses()	giver et array af de indre klasser (og interfaces), der er public i klassen

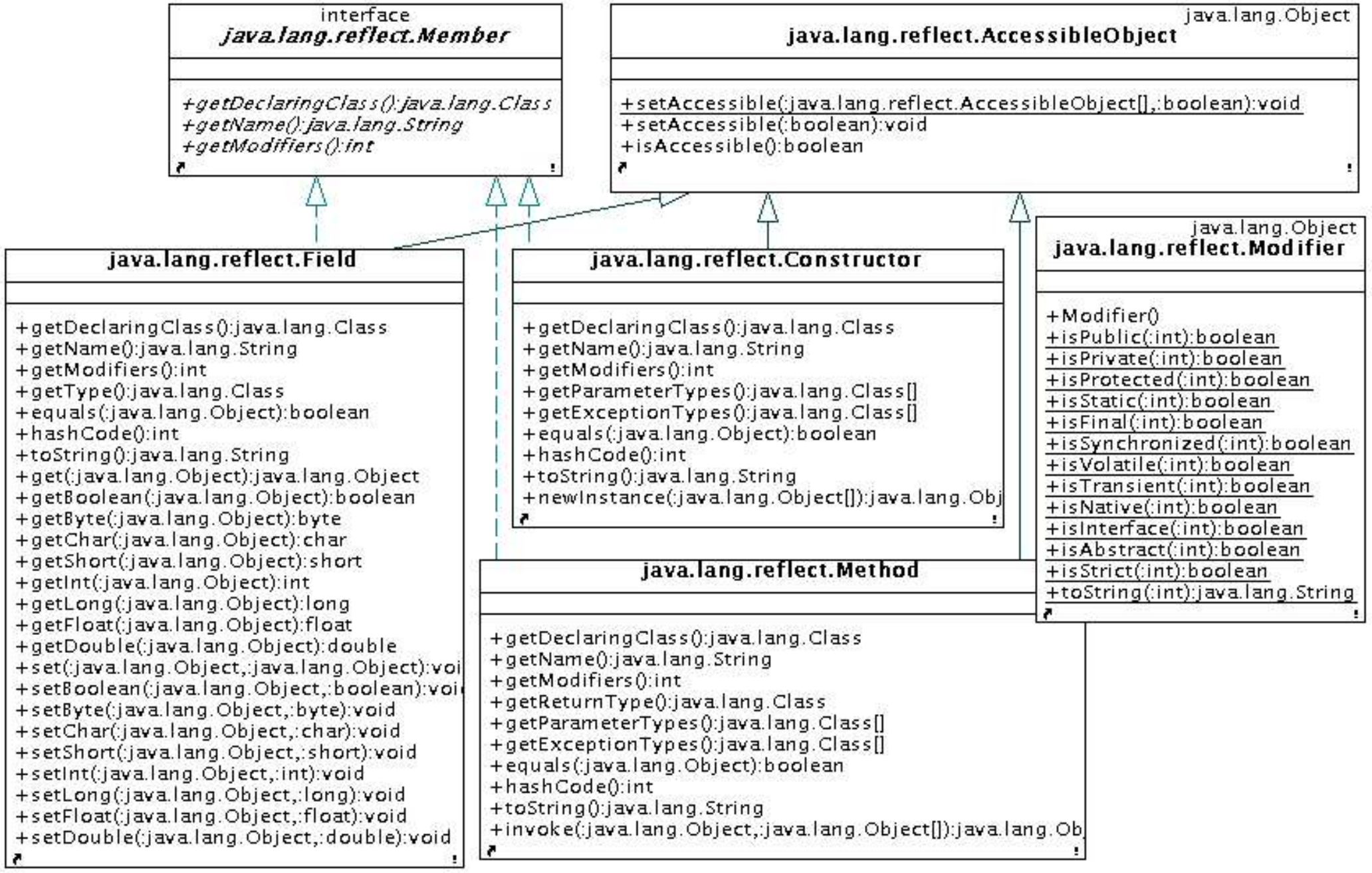
```
import java.lang.refle
import java.awt.*;

public class FindMetod
{
    public static void
    {
       Object o = new Bu

       // Find klassen
       Class klasse = o.getClass();
       System.out.println("Klassen navn er: "+klasse.getName());

       Method[] metoder = klasse.getMethods();
       for (int i=0; i<metoder.length; i++)
       {
          Method m = metoder[i];
          System.out.print("Metode "+m.getName());
          System.out.print(" har returtype: "+m.getReturnType().getName());
          Class[] parametertyper = m.getParameterTypes();
          System.out.print(" og parametertyper:");
          for (int j=0; j<parametertyper.length; j++)
             System.out.print(" " + parametertyper[j].getName());
       }
       System.out.println();
    }
}
```

```
Klassen navn er: java.awt.Button
...
Metode: notify har returtype: void
Metode: notifyAll har returtype: void
Metode: toString har returtype: java.lang.String
...
Metode: getLabel har returtype: java.lang.String
Metode: setLabel har returtype: void og parametertyper:
java.lang.String
...
```

```
import java.lang.reflect.*;
import javax.swing.*;
import java.beans.*;

public class Boenneintrospek
{
    public static void main(
    {
      Object objekt = new JE
      Class klasse = objekt.
      BeanInfo bønneinfo = Introspector.getBeanInfo(klasse);
      PropertyDescriptor egenskaber[] = bønneinfo.getPropertyDescriptors();

      for (int i=0; i<egenskaber.length; i++)
      {
        PropertyDescriptor e = egenskaber[i];

        System.out.print(e.getName()+": "+e.getShortDescription());

        Method læsemetode = e.getReadMethod();
        if (læsemetode != null)
        {
          Object[] tomParameterliste = {};
          Object værdi = læsemetode.invoke(objekt,tomParameterliste);
          System.out.print(" (værdi="+værdi+")");
        }

        // sæt egenskaben til true, hvis den kan sættes og er af type boolean
        Method skrivemetode = e.getWriteMethod();
        if (skrivemetode!=null && e.getPropertyType()==java.lang.Boolean.TYPE)
        {
          Boolean[] parameterlisteMedTRUE = { Boolean.TRUE };
          skrivemetode.invoke(objekt, parameterlisteMedTRUE ); // sæt egenskab
        }
        System.out.println();
      }
    }
}
```

```
inputMap: inputMap (værdi=javax.swing.InputMap@6e3d60)
insets: insets (værdi=java.awt.Insets[top=5,left=17,bottom=5,right=17])
alignmentY: The preferred vertical alignment of the component. (værdi=0.5)
alignmentX: The preferred horizontal alignment of the component. (værdi=0.0)
toolTipText: The text to display in a tool tip. (værdi=null)
mnemonic: the keyboard character mnemonic (værdi=0)
verticalAlignment: The vertical alignment of the icon and text. (værdi=0)
defaultButton: Whether or not this button is the default button (værdi=false)
rolloverEnabled: Whether rollover effects should be enabled. (værdi=false)
horizontalAlignment: The horizontal alignment of the icon and text. (værdi=0)
borderPainted: Whether the border should be painted. (værdi=true)
```

Pakken java.lang.reflect



Egne rutiner til at hente/gemme data

- Det er ret nemt at skrive 'serialiseringsrutiner' til at indlæse/gemme data i eget format
- Eksempel: Læse data fra fil og gemme i Boks-objekt
  - Kræver at objektvariablerne er public

```
String navn = bidder.nextToken();
double værdi = Double.parseDouble(bidder.nextToken());

if (navn.equals("længde")) boks.længde = værdi;
if (navn.equals("bredde")) boks.bredde = værdi;
if (navn.equals("højde")) boks.højde = værdi;
// ... osv. Osv osv...
```

```
String navn = bidder.nextToken();
double værdi = Double.parseDouble(bidder.nextToken());

// smartere: Udnyt variabelnavn=navn i datafil!
try {
    boks.getClass().getField(navn).setDouble(boks,værdi);
} catch (Exception e) {
    e.printStackTrace();
}
```

Egne rutiner til at redigere i data

```
import java.lang.reflect.Field;

import javax.swing.*;
import javax.swing.table.AbstractTableModel;

public class RedigerObjekt extends JPanel {
    JScrollPane jScrollPane1 = new JScrollPane();
    JTable jTable1 = new JTable();

    private Object objektet;
    private Class klassen;
    private Field[] felter;

    public void setObjekt(Object objekt) {
       this.objektet = objekt;
       klassen = objektet.getClass();
       felter = klassen.getDeclaredFields();
       jTable1.setModel(new ObjektTableModelAdapter());
    }

    public Object getObjekt() {
       return objektet;
    }


    public RedigerObjekt() {
       try {
          jbInit();
       } catch (Exception ex) {
          ex.printStackTrace();
       }
    }
```

Egne rutiner til at redigere i data

```
class ObjektTableModelAdapter
   extends AbstractTableModel
{
   public int getColumnCount()
   {
      return 2;
   }

   public int getRowCount()
   {
      return felter.length;
   }

   public Object getValueAt(int ræk, int kol)
   {
      if (kol==0) return felter[ræk].getName();

      try {
         return felter[ræk].get(objektet);
      }
      catch (Exception ex) {
         ex.printStackTrace();
         return "???";
      }
   }

   public String getColumnName(int kol)
   {
      if (kol==0) return "felt";
      else return "værdi";
   }
```

```
public boolean isCellEditable(int ræk, int kol
{
   if (kol==0) return false;
   else return true;
}

public void setValueAt(Object værdi, int ræk,
   System.out.println("setValueAt(" + værdi);
   try {
      Field f = felter[ræk];
      Class t = f.getType();
      if (t == Double.TYPE) {
         f.setDouble(objektet,
          Double.parseDouble(værdi.toString()));
      } else if (t == Integer.TYPE) {
         f.setInt(objektet,
          Integer.parseInt(værdi.toString()));
      } else if (t == String.class) {
         f.set(objektet, værdi);
      } else JOptionPane.showMessageDialog(null
          f + " ej understøttet");
   } catch (Exception ex) {
      ex.printStackTrace();
      JOptionPane.showMessageDialog(null, ex);
   }
}
```

```
}
```

Metoder/variabler der ikke er public

- getMethod() -> get*Declared*Method()
  getField() -> get*Declared*Field()
  ...
- setAccessible(true) for at slå adgangskontrol fra
  - Kræver at security-manageren tillader det!

```
String navn = bidder.nextToken();
double værdi = Double.parseDouble(bidder.nextToken());

// sætter også private variabler!
try {
    java.lang.reflect.Field f;
    f = boks.getClass().getDeclaredField(navn);
    f.setAccessible(true);
    f.setDouble(boks,værdi);
} catch (Exception e) {
    e.printStackTrace();
}
```

JAR-filer

- JAR-fil er en ZIP-fil med klasser
  - jar cf program.jar BenytPakker.class minPakke
  - zip -r program.jar BenytPakker.class minPakke
- Værktøjet kan lave den for en!

- Oprettelse af eksekverbare JAR-filer
  - JAR-fil med META-INF/MANIFEST.MF
    - Her er klassen med main-metoden angivet

```
Manifest-Version: 1.0
Main-Class: BenytPakker
```

```
Manifest-Version: 1.0
Created-By: 1.4.2 (Sun Microsystems Inc.)
Main-Class: java2d.Java2Demo
```

    - Kan aktiveres ved dobbeltklik (ligesom .exe-fil) eller med java -jar program.jar

Optimering af programmer

- Programmér først – optimér bagefter
  - Identificere flaskehalsene og optimer *kun* disse dele af koden
    - Spild af tid (og overblik) at optimere unødvendigt
- Optimering af hukommelsesforbrug
  - Vær bevidst om hvad objekter fylder
    - Mange små objekter kan være et problem
      - Brug et eller flere array
    - Flerdimensionelle array – er faktisk array af array (et array er et objekt)
      - Størrelse af new byte[1][1000000]: 1MB (+ array-objekt á 16 byte)
      - Størrelse af new byte[1000000][1]: 17MB (=1000000 array á 16+1 byte!)
- CPU-forbrug
  - Undgå:
    - at oprette mange objekter som skal nedlægges igen
      - Vær især opmærksom på String (og andre uforanderlige objekter)
    - at oprette mange tråde
    - at kaste og fange mange undtagelser
    - synkroniserede blokke/metoder
    - mange (evt. anonyme) klasser
    - introspektion

Optimering af programmer

```
// Demonstrerer hastighedsforskellen mellem String og StringBuffer
// ved sammensætning af mange strenge
public class HastighedsforskelMellemStringOgStringBuffer
{
    public static void main (String[] arg)
    {
       long tid1 = System.currentTimeMillis();

       String s = "";
       for (int i=0; i<10000; i++) s = s + "x";   // her oprettes 10000 objekter

       long tid2 = System.currentTimeMillis();
       System.out.println("Antal sekunder med String: "+ (tid2-tid1)*0.001 );

       StringBuffer sb = new StringBuffer(10000); // reservér plads til 10000 tegn
       for (int i=0; i<10000; i++) sb.append("x");// her ændres i det samme objekt
       String s2 = sb.toString();

       long tid3 = System.currentTimeMillis();
       System.out.println("Antal sek med StringBuffer: "+ (tid3-tid2)*0.001 );
    }
}
```

```
Antal sekunder med String: 3.432
Antal sek med StringBuffer: 0.021
```

Optimering: Java versus C/C++

- Hukommelsesforbrug
  - Java-programmer fylder mere end C-programmer
- CPU-forbrug
  - Java 'hotspot' JIT-oversætter
    - 'hotspot' – kun kode der løbes igennem mange gange
    - JIT – Just In Time – oversætter bytekode til maskinkode
- Java kan optimere efter hvordan programmet rent faktisk kører det meste af tiden
  - Java i visse tilfælde hurtigere end C!
- Opstart af Java-program er (altid) langsommere end et C-program

Evt.: Optimeringsværktøjer (Borland Optimizelt)