

## Lagring av objekt

De två sista typerna är enkla variabler vilka bara är att lagra. förnamn och efternamn däremot är referensvariabler och därför är det värdelöst att spara dem ty de objekt som refereras till lär inte ligga på samma plats i minnet nästa gång programmet körs.

För att lyckas med att spara undan alltihop implementerar man Serializable.

Serializable är dock tomt!

### Exempel

Skriv person-objekt till fil.

### Exempel

Läs in person-objekt från fil.

readObject returnerar en referens av typen Object varför vi måste göra en typomvandling.

Det hela hade kunnat göras ännu enklare om vi i första programmet hade haft en lista med personer. Då hade vi kunnat lägga ut hela listan i filen och på samma sätt läsa in hela listan när vi vill söka.

## Model View Controller

Om en egenskap lämpligen beskrivs mha en Array kan det vara ytterst lämpligt att använda IndexedPropertyChangeEvent.

JavaBeans är ett exempel på hur man lämpligen använder sig av MVC.

### Lagring av objekt i binärfiler

```
import java.io.*;

public class Person implements Serializable {
    String förnamn;
    String efternamn;
    int föddår;
    boolean singel = true;
}

import javax.swing.*;
import java.io.*;

public class SparaPersoner {
    public static void main(String[] arg) throws IOException {
        ObjectOutputStream utström = new ObjectOutputStream(
            new FileOutputStream("personfil.data"));

        while (true) {
            String s = JOptionPane.showInputDialog("Förnamn? ");
            if (s == null)
                break;
            Person p = new Person(); // skapa ett nytt objekt!!
            p.förnamn = s;
            p.etternamn = JOptionPane.showInputDialog("Efternamn? ");
            s = JOptionPane.showInputDialog("Födelseår? ");
            p.föddår = Integer.parseInt(s);
            int knappNr = JOptionPane.showConfirmDialog(null, "Singel?");
            p.singel = knappNr == 0;
            utström.writeObject(p);
        }
        utström.close();
    }
}

import javax.swing.*;
import java.io.*;

public class SökPersoner {
    public static void main(String[] arg) throws Exception {
        ObjectInputStream inström = new ObjectInputStream(
            new FileInputStream("personfil.data"));

        String sökt = JOptionPane.showInputDialog(
            ("Den sökta personens efternamn?"));

        Person p;
        while (true) {
            try {
                p = (Person) inström.readObject();
            }
            catch (EOFException e) {
                break; // filen är slut
            }

            if (p.etternamn.equals(sökt)) {
                String s = p.förnamn + " " + p.etternamn +
                    ", född år " + p.föddår + ", är ";
                if (!p.singel)
                    s = s + " inte ";
                s = s + " singel";
                JOptionPane.showMessageDialog(null, s);
            }
        }
        inström.close();
    }
}
```

### Properties

- Ett objekt kan ha vissa **egenskaper** (som beskrivs med instansvariabler).
- En enkel egenskap *X* avläses och sätts med metoderna `getX()` och `setX(värde)`.
- En indexerad egenskap (från t.ex. en array) avläses och sätts med metoderna `getX(i)` och `setX(i, värde)`
- När en enkel egenskap ändras kan objektet generera en händelse av typen `PropertyChangeEvent`.
- När en indexerad egenskap ändras kan objektet generera en händelse av typen `IndexedPropertyChangeEvent`, en subclass till `PropertyChangeEvent`.
- Exempel på detta är standardklasserna i `Swing`.
- Händelser kan fångas upp av lyssnare av typen `PropertyChangeListener` som har en metod med namnet `propertyChanged`.

Användbara metoder som kan anropas i `propertyChanged`:

- `getPropertyName`
- `getNewValue`
- `getOldValue`
- `getIndex` (för händelser av typen `IndexedPropertyChangeEvent`)

### JavaBeans

Ett standardmönster:

- Har parameterlös konstruktor
- Implementerar `Serializable`
- Har egenskaper
- Har metoder `setX`, `getX`, `isX`
- Kan ha indexerade metoder `setX`, `getX`, `isX`

En JavaBean kan registrera lyssnare för händelser av typen `PropertyChangeListener`. Kan t.ex. implementera följande gränssnitt:

```
import java.beans.*;
import java.io.*;

public interface MyBeanModel extends Serializable {
    void addPropertyChangeListener(PropertyChangeListener l);
    void removePropertyChangeListener(PropertyChangeListener l);
}
```

Exempel

Observer Pattern

Exempel på en vy.

MVC

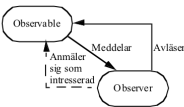
Exempel på Controller.

- Problem i praktiken:
- Komponenter i vyn används ofta även för inmatning (användaren klickar t.ex.)
  - Svårt att separera Controllern från Vyn
  - I Swing används s.k. UI-delegates som fungerar både som vy och controller.

```
import java.beans.*;
public class MyBeanClass implements MyBeanModel {
    private final PropertyChangeSupport p=new PropertyChangeSupport(this);
    public void addPropertyChangeListener(PropertyChangeListener l) {
        p.addPropertyChangeListener(l);
    }
    public void removePropertyChangeListener(PropertyChangeListener l) {
        p.removePropertyChangeListener(l);
    }
    private double value;
    private double[] arr;
    public MyBeanClass () { ... } // konstruktor
    public double getValue() { return value; }
    public void setValue (double newValue) {
        double oldValue = value;
        value = newValue;
        p.firePropertyChange("value", oldValue, newValue);
    }
    public double getArr(int index) { return arr[index]; }
    public void setArr(int index, double newElem) {
        double oldElem = arr[index];
        arr[index] = newElem;
        p.fireIndexedPropertyChange("arr", index, oldElem, newElem);
    }
}
```

Observer pattern

Ett vanligt s.k. designmönster

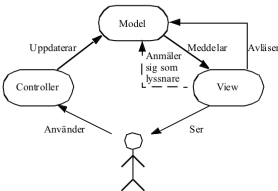


Kallas **Model-View** för grafiska komponenter.  
Modellen kan göras som en JavaBean.

```
import java.beans.*;
import javax.swing.*;
public class MyView extends JPanel
    implements PropertyChangeListener {
    private MyBeanModel model;
    public MyView(MyBeanModel m) {
        model = m;
        model.addPropertyChangeListener(this);
    }
    // utplacering av grafiska komponenter
    ...
    public void propertyChange(PropertyChangeEvent e) {
        double v = model.getValue();
        // ändra grafiska komponenter
        ...
    }
}
```

Model-View-Controller

Controllern tar emot signaler utifrån och uppdaterar modellen:



```
import javax.swing.*;
import java.awt.event.*;
public class MyController extends JPanel
    implements ActionListener {
    private MyBeanModel model;
    private JTextField f = new JTextField(5);
    public MyController (MyBeanModel m) {
        model = m;
        add(f);
        f.addActionListener(this);
    }
    public void actionPerformed(ActionEvent e) {
        if (model != null)
            model.setValue(Integer.parseInt(f.getText()));
    }
}
```