JAVABEANS ÜBUNGEN

In diesem Kapitel:

- Das Bean Development Kit (BDK)
  - *Installation und Testen des BDKs*
- Testen und Modifizieren einer JavaBean
  - *Textausgabe einer Bean in einem Applet*
- Modifikation und Erweiterung einer JavaBean
  - *verbesserte Textausgabe*
  - *variable Texteingabe*
  - *Text-Animation*
- Bean Inspektion und BeanInfo
  - *Definition der BeanInfo Klasse*
  - *Implementation einer beanInfo*
- Zusammenfassen mehrerer Beans
  - *Bean Programm*
  - *Listener und Event*
- Ressourcen
  - *Einsatz von Ressourcen in Beans*
- Customizer
  - *Bau und Einsatz eines Customizers*
- Reflection API
  - *Einsatz des reflection APIs zur Bestimmung der Metadaten*
- Signaturen
  - *Vertrauenswürdige Beans*
  - *X.509 Public / Private Keys*

JavaBeans

-

Übungen

1.1. Einleitung

1.1.1. Übergeordnete Kursziele

Das übergeordnete Ziel dieser Einführung in die JavaBeans Technologie ist es, Ihnen zusammen mit praktischen Beispielen oder Übungen folgende Technologien näher zu bringen:

- die JavaBeans Architekture
- das Beans Event Modell
- Introspektion - Abfrage des Inhaltes einer JavaBean
- kreieren von Bean Komponenten
- Anpassen von Beans
- Persistenz zum Speichern und Aktivieren von Beans
- die BDK BeanBox

1.1.1.1. Kurs-Voraussetzungen

Dieser Kurs setzt voraus, dass Sie mit den Objekt-orientierten Konzepten vertraut sind und AWT, Threads und natürlich die Java Grundlagen kennen.

1.1.1.2. Format des Kurses

Dieser Kurs besteht aus Übungen und etwas Theorie. Vorallem sollten Sie aber einen praktischen Einblick in die JavaBeans gewinnen. Dies ist nur möglich, falls Sie die Übungen mindestens anschauen. Alle Übungen enthalten auch Musterlösungen, so dass Sie mindestens eine Ahnung über die praktische Umsetzung erhalten.

1.1.1.3. Vorgehensweise beim Durcharbeiten

Jede Person hat ihren eigenen Lernstil. Sie können mit den Übungen beginnen und dann zur Theorie wechseln. Sie können aber auch in diesem Theorieteil starten und dann zu den Übungen übergehen.

1.1.2. Generelle Bemerkungen zu den Übungen

Eine Übung besteht aus der Aufgabenstellung und Hilfestellungen. Einige der Übungen können Sie eventuell selbständig lösen; bei anderen werden Sie Hilfe oder Lösungshinweise benötigen, beispielsweise weil die Aufgabenstellung für Sie zu unverständlich ist!Sie können Sich aber auch einfach die Musterlösung ansehen, sofern eine existiert (beispielsweise bei der Installation der Software kann man kaum eine Musterlösung erwarten: die Installation geschieht auf Ihrem Rechner). Die einzelnen Übungen bauen aufeinander auf. Aber da Sie in der Regel eine Musterlösung haben, sollte es Ihnen trotzdem möglich sein, den Übungen zu folgen.

1.1.3. Die Anatomie einer Übung

Jede Übung enthält eine Liste jener Übungen, welche Sie bereits gelöst haben sollten, um die aktuelle Übung verstehen zu können. Zudem finden Sie Rahmenprogramme, welche Sie als Basis für die eigene Lösung verwenden können. Falls vorhanden, werden Sie auch noch Links zu den API Seiten finden, damit Sie eventuell benötigtes Hintergrundwissen ergänzen oder nachschlagen können.Jede Übung enthält eine Liste der Lernziele, also der Ziele, die Sie hoffentlich erreichen. Die Hilfestellungen finden Sie gleich anschliessend an die Fragestellungen, in der Regel Schritt für Schritt.

Sie finden also pro Übung folgende Informationen:

- **Hilfen**
  mit Hinweisen zur Lösung der gestellten Aufgabe;
  eine Lösungsskizze mit Kommentaren und allem drum und dran;
- **Musterlösung**
  Die Musterlösung enthält auch die Applet Tags und alle benötigten Dateien.
- **API Dokumentation**
  Links zu den Online Dokumentation bei Sun

1.1.4. Design Ziele für die Übungen

Es gibt drei Arten von Übungen:

- **"Blank-Screen"** Übungen
  Sie finden einen leeren Bildschirm vor ("Blank Screen") und müssen die gesamte Lösung selber erarbeiten.
- **Erweiterungen**
  Sie erweitern eine bereits bestehende Lösung
- **Korrektur**
  Sie korrigieren das Verhalten eines bestehenden Programms.

Um Ihnen das Erlernen der Servlet Techniken zu erleichtern, werden in den Übungen in der Regel nur jene Teile zu erschaffen sein, welche spezifisch für Servlets sind. Alle irrelevanten und komplexen Bereiche, die Sie eventuell zur Lösung benötigen, werden Ihnen zur Verfügung gestellt.

In der Regel werden die Übungen mit Hilfe eines Web Servers / des Webs durchgeführt. In den Fällen, in denen Sicherheitsprobleme auftauchen könnten, werden die Beispiele lokal ausgeführt.

In dieser Kurseinheit lernen Sie praktisch, wie man Servlets erstellt und einsetzen kann. Nach dem Durcharbeiten dieser Kurseinheit sollten Sie wissen, wie man Servlets erstellt und in Web Servern einbaut. Sie werden auch lernen, wie Client Applets mit Servlets kommunizieren können, um Server oder Datenbank Informationen abzufragen. Darauf aufbauend bleibt es Ihnen überlassen, ausgefeilte Anwendungen zu entwickeln und die Tiefen der Technologie auszuloten....

1.2. Übungen

In diesen Übungen wollen wir nun endlich konkrete JavaBeans entwickeln. Als Basis verwenden wir zuerst das BDK, das Bean Development Kit von Sun. Im Anhang sehen Sie wie man Beans mit dem JBuilder entwickelt.

Einige der Übungen zeigen, wie man Beans baut, andere zeigen, wie man auf deren Metadaten zugreift. Zudem lernen Sie, wie man eigene beans in das BDK einbinden kann.

Natürlich lernen Sie auch, wie man mit den Reflection API im Bean Umfeld arbeitet und Beans signiert werden können, um deren Sicherheit zu erhöhen.

Insgesamt werden Sie neun Übungen durcharbeiten.

1. **BDK Einführung**
   In dieser Übung lernen Sie das BDK kennen, die Bean Box und den Test Container. Zudem sehen Sie bereits an einigen vorgefertigten Beans, wie solche funktionieren und welche Properties und Customizer vorhanden sind.

   Sie spielen mit dem Jonglierer 'Duke' und bauen selber aus bestehenden Beans neue zusammen.

   Lernziele:
   - Sie wissen was das BDK ist
   - Sie können abschätzen zu was das BDK gebraucht werden kann
   - Sie kennen einige Beispiel-Beans
   - Sie lernen, wie einzelne Beans zusammengesetzt werden können
   - Sie lernen, wie man Bean Properties verändert.

2. **Kreieren Ihrer ersten Bean**
   In dieser Übung bauen Sie endlich Ihre eigene erste Bean.

   Lernziele:
   - Sie definieren eine Bean
   - Sie kreieren eine eigene Bean!

3. **Erweitern einer bestehenden JavaBean**
   Aufbauend auf der vorherigen Übung bauen Sie die JavaBean aus. An Stelle einer fixen Text Area verwenden Sie dazu rollbare Flächen, Events und constrained Properties.

   Als Basis erhalten Sie ein Rahmenprogramm, welches Sie zu einer Bean machen müssen.

   Lernziele:
   - Sie lernen, wie man Bean Komponenten kreiert
   - Sie arbeiten mit constrained Properties
   - Sie definieren und arbeiten mit Source Events
   - Sie stellen erste Überlegungen an, um Persistenz einzubauen

4. **BeanInfos und Bean Inspektion**
   Mit Hilfe von Reflection und BeanInfo bauen Sie Zusatzinformationen in Ihre JavaBeans ein. Zusätzlich verwenden Sie das Reflection API. Sie verwenden zusätzlich Methoden, welche es Ihnen gestatten, die Informationen zu bestimmen und auszuwerten (Properties, Event Listener).

   Ihre Aufgabe besteht darin, die Inspektionsmethode des Rahmenprogramms zu ergänzen, zu vervollständigen, um alle Namen und Werte der Properties für alle Event Listener der Bohne zu bestimmen.

   Lernziele:
   - Sie lernen, wie man eine Bohne inspiziert, mit Hilfe der BeanInfo
   - Sie bestimmen Events und Properties einer Bohne
   - Sie rufen Methoden auf, ohne deren Namen zu kennen.

5. **Packaging**
   In dieser Übung packen Sie mehrere Beans zusammen und bilden ein Java Archiv, welches auch mit der BeanBox verwendet werden kann. Zusätzlich schreiben Sie eine BeanInfo Klasse.

   Lernziele:
   - Sie lernen Beans zu verpacken
   - Sie lernen, wie man eine Manifest Datei kreiert
   - Sie lernen, wie eine Bohne in die BeanBox / ToolBox geladen wird.

6. **Java Ressource Dateien**
   Applets verfügen über hilfreiche Methoden, wie `getCodeBase()`, `getDocumentBase()` oder `getParameter()`.
   Beans sind aber selbständige Software Komponenten und benötigen einen anderen Mechanismus, das Java Resource File. .
   Diese Datei kann entweder im Archiv mitverpackt oder in der lokalen CLASSPATH gestellt werden.

   Lernziele
   - Sie lernen die Bedeutung von Ressourcen für JavaBeans kennen
   - Sie lernen Resource Files zu laden.

7. **Customizer**
   Nun ergänzen wir eine bestehende JavaBean durch einen Customizer. Zudem kreieren wir eine Applikation, mit deren Hilfe Beans gespeichert und wieder geladen werden können.

   Lernziele:
   - Sie lernen wie man einen Bean Customizer schreibt
   - Sie lernen wie man Beans serialisiert und speichert bzw. wieder lädt.
   - Sie lernen mit Customizern umzugehen.

8. **Relektive Programmierung - der Einsatz des Reflection APIs**
   Diese Übung ist optional.
   Falls Sie je komplexere Anwendungen schreiben wollen, sollten Sie sich mit den Details des Reflection APIs und der Introspection für JavaBeans vertraut machen.

   Lernziele:
   - Sie lernen den Einsatz des Reflection APIs in JavaBeans kennen
   - Sie kennen den Einsatz der Introspection für die Bestimmung der Beans Eigenschaften ohne BeanInfo.

9. **Signaturen und JavaBeans**
   Auch diese Übung ist optional.
   Falls Sie Beans verkaufen wollen, sollten Sie sich darüber bewusst sein, dass die Security wegen der Download Möglichkeiten von JavaBeans über das Internet eine grosse Rolle spielt. Für jeden einzelnen Benutzer wird eine eindeutige Kennung, ein Schlüssel benötigt, beispielsweise ein public X.509 Schlüssel. Dieser wird in irgend einer Datenbank verwaltet. (Sie könnten auch komplexere Sicherheitsmechanismen mit dem Security API aufbauen).

   In dieser Übung werden Sie durch das Signieren eines Applets geführt, Schritt für Schritt! Als Basis verwenden Sie dazu das Ergebnis aus der vorigen Übung.

   Lernziele:
   - Sie lernen JAR Dateien / Beans zu signieren
   - Sie lernen privilegierte Beans zu kreieren.

1.3. BDK Einführung

In dieser Übung lernen Sie das BDK kennen, die Bean Box und den Test Container. Zudem sehen Sie bereits an einigen vorgefertigten Beans, wie solche funktionieren und welche Properties und Customizer vorhanden sind.

Sie spielen mit dem Jonglierer 'Duke' und bauen selber aus bestehenden Beans neue zusammen.

1.3.1. Lernziele

- Sie wissen was das BDK ist
- Sie können abschätzen zu was das BDK gebraucht werden kann
- Sie kennen einige Beispiel-Beans
- Sie lernen, wie einzelne Beans zusammengesetzt werden können
- Sie lernen, wie man Bean Properties verändert.

Als erstes müssen Sie das BDK ab dem Sun Web Site herunterladen:
http://java.sun.com/products/javabeans/

Hier noch ein Hinweis: sollten Sie beim Installieren die fehlermeldung erhalten, dass Sie keine oder eine falsche Java VM installiert haben, diese Aussage aber nicht zutrifft, dürfte dies am Installer liegen. Dieser verlangt, dass java.exe im Pfad vorhanden ist. Sie können dies leicht testen in dem Sie in einem DOS Fenster java eintippen. Falls Sie nicht die Standard Usage Meldungen der JVM erhalten, ist die Datei nicht im Pfad definiert, der Installer wird also Probleme haben.

1.3.2. Ihre Aufgaben:

1. Öffnen Sie ein DOS Fenster und wechseln Sie ins BeanBox Verzeichnis des BDKs
   In meiner Installation ist dies folgendes Verzeichnis:
   `D:\BDK1.1\beanbox>`

   Passen Sie das Skript `run.bat` an:
   ```
   @echo off
   if "%OS%" == "Windows_NT" setlocal
   set CLASSPATH=classes;..\lib\methodtracer.jar;..\infobus.jar
   %JAVA_HOME%\bin\javaw sun.beanbox.BeanBoxFrame
   ```

   Die Änderung, welche ich bei meiner Installation machen musste, sehen Sie oben fett dargestellt.`javaw` verwende ich, um nach dem Starten das DOS Fenster schliessen zu können. Diese Variante ist aber weiter unten nicht brauchbar, da bei der Introspection die Ausgabe in `System.out` geschieht d.h. ins DOS Fenster.
   Starten Sie damit die BeanBox.

2. Die BeanBox besteht aus mehreren Teilen: der ToolBox, der BeanBox, dem Property Sheet und dem Method Tracer. Die letzten drei sind nur sichtbar, falls Sie im design Modus sind. Sie können den Modus umschalten, indem Sie in der BeanBox unter `View` umschalten. Für die folgenden Übungen sollten Sie im design Modus sein, also alle Fenster sehen.

3. Das aktive / aktuelle Bean ist in der BeanBox daran erkennbar, dass sie umrandet ist. Im PropertySheet sehen Sie alle Eigenschaften der Bean. Um Bean Eigenschaften zu ändern, müssen Sie einfach im Property Sheet Anpassungen vornehmen.
   Wenn Sie auf eine Eigenschaft doppelklicken, wird der Property Editor für diese Eigenschaft angezeigt (PopUp), beispielsweise eine Farbpalette oder der Font Editor, falls Sie auf den Text klicken und ein Editor hinterlegt wurde.

4. Um irgend etwas in der BeanBox zu plazieren, selektieren Sie in der ToolBox eine Bohne (Sie sehen ein Kreuz, klicken auf den Text oder das Icon) und plazieren diese anschliessend in der BeanBox. Sie können die Bohne immer noch verschieben und mit den oben erwähnten Editoren verändern, anpassen.

   In der aktuellen Version der ToolBox sind bereits einige Beans vorhanden, zum Beispiel Buttons und ein Molekül-Viewer, ein Sorter-Bean und ein Event Monitor sowie einige weitere Beans. Plazieren Sie einige dieser Beans in die BeanBox. Rotieren Sie das Molekül: es ist alles Live! Beim SortedBean müssen Sie noch einmal auf die Bohne klicken, um sie zu starten. Durch Selektion und 'Cut' können Sie die Beans jederzeit aus der beanBox entfernen.

5. Als erstes betrachten wir nun den `ExplicitButton` (zweite Bohne im Toolkit) genauer. Als erstes löschen Sie alle Beans aus der BeanBox und plazieren lediglich die `ExplicitButton` Bean in der Box. Dieser Knopf besitzt einen Customizer! Sie können ihn sichtbar machen, indem Sie ihn im BeanBox Menü unter 'Edit' sichtbar machen.

   Als erstes wollen wir allerdings einen report generieren, also Introspection anwenden. Dazu verwende ich folgendes modifiziertes Skript:

```
@echo off
if "%OS%" == "Windows_NT" setlocal
set CLASSPATH=classes;..\lib\methodtracer.jar;..\infobus.jar
%JAVA_HOME%\bin\javaw sun.beanbox.BeanBoxFrame > Report.txt
```

   um die Ausgabe ins (wegen javaw) nicht mehr vorhandene Fenster in eine Datei umzuleiten.Eine vollständige Beispielausgabe finden Sie auf dem Server / der CD. Hier ein Auszug aus der Ausgabe:

```
Properties:
    foreground        class java.awt.Color
        getForeground/setForeground
    label             class java.lang.String
        getLabel/setLabel
    background        class java.awt.Color
        getBackground/setBackground
    font              class java.awt.Font
        getFont/setFont
```

   Diese vier Properties entsprechen jenen auf dem Property Sheet. Zudem sehen Sie die zur Verfügung stehenden Methoden und können daraus herleiten, ob es sich um `read-only, write-only` oder um `read/write` Eigenschaften handelt.

   Analog verhält es sich mit den Events:

jedes Event ist eine Unterklasse von `EventObject`.

```
Event sets:
    actionPerformed
        addActionListener/removeActionListener
        actionPerformed
    propertyChange
        addPropertyChangeListener/removePropertyChangeListener
        propertyChange
```

Sie finden die selben Events auch unter dem *Edit* Menü in der BeanBox unter 'Events'

Plazieren Sie jetzt noch die `ChangeReporter` Bean in die BeanBox. Nun können wir mit den Events un den Bohnen spielen:

immer wenn der `ExplicitButton` geändert wird, soll der `ChangeReporter` benachrichtigt werden.

Das lässt sich ohne grosse Programmierung realisieren:
- selektieren Sie den ExplicitButton und öffnen Sie das Edit Menü.
- selektieren Sie nun unter Events 'bound property change' und daraus 'propertyChange'.
- jetzt sollten Sie vor Ihrer Maus eine rote "RubberLine" sehen.
- ziehen Sie diese zum ChangeReporter Textfeld, eigentlich auf den nicht sichtbaren selektierten Rand. Die rote Linie verschwindet und ein Menü erscheint, der EventTargetDialog, in dem Sie alle öffentlichen Methoden des ChangeReporters sehen.
- wählen Sie die Methode '`reportChange()`' aus (zweite von oben)
- klicken Sie auf OK. Ein DOS Fenster öffnet sich kurz Ihre Änderung wird fixiert!

Nun können Sie das Ganze testen: ändern Sie irgend eine Eigenschaft im ExplicitButton und schauen Sie sich die Angabe im Textfeld an!

Beispiele:
ändern Sie den Text / das Label (Beispielausgabe: `label := Hallo`)
ändern Sie die Farben (Beispielausgabe: `background := java.awt.Color[r=255,g=255,b=0]`)

6. Nun aktivieren wir eine Verbindung zwischen dem Button und dem Textfeld, falls der Knopf gedrückt wird:

   dazu müssen wir die Methode 'actionPerformed()' mit (Events -> buttonPush()) mit dem Textfeld verbinden und dort eine Methode auswählen.

   Damit sehen Sie immer wenn der Button angeklickt wird beispielsweise eine Meldung im Textfeld.

7. Als weitere Bean plazieren wir nun die `JellyBean` in die BeanBox.
   Diese verwenden wir, um Properties verschiedener Beans miteinander zu koppeln: wenn sich die Eigenschaft der einen Bean ändert, propagiert die Änderung zu einer weiteren Bohne und bewirkt dort eine passende veränderung.

   Und so müssen Sie vorgehen:
   0) selektieren Sie die JellyBean
   1) selektieren Sie das 'Edit' Menü
   2) wählen Sie 'BindProperty', dann 'color' aus der Liste und klicken Sie auf OK.
   3) nun sehen Sie die rote RubberLine.
   4) Verbinden Sie nun den JellyBean mit dem ExplicitButton.
   5) wählen Sie 'background' aus
   6) testen Sie die Verbindung: wenn Sie die Farbe im JellyBean verändern, ändert sich auch die Farbe im Button. Zudem wird ein Ereignis ausgelöst. Dieses wird in der Textfläche sichtbar.

8. Nun löschen wir alles: clear im Datei Menü.löscht alle Beans aus der BeanBox.
   Die BDK warnt Sie nicht: alle Änderungen gehen verloren. Wenn Ihnen dies nicht gefällt, können Sie den Quellcode von BDK, der mitgeliefert wird, anpassen.

9. Unsere letzte Aufgabe starten wir mit der `Juggler` Bean, der Bohne, welche den jonglierenden Duke zeigt. Unterhalb der Bean plazieren wir zwei `ExplicitButton` Beans.

   Benennen Sie die Buttons: der eine Stop, der andere Start.
   Verbinden Sie jetzt, analog zu oben, diese Buttons mit Methoden der `Juggler` Bean:
   - 'Stop' ruft die Methode '`stopjuggling()`' der `Juggler` Bean auf:
   - 'Start' ruft die Methode '`startjuggling()`' der `Juggler` Bean auf.

   Sie haben nun ein voll funktionsfähiges Programm!

   Speichern Sie das Ganze ab, beispielsweise unter 'duke.ser' im 'File' Menü. Die Dateierweiterung 'ser' zeigt an, dass es sich um ein serialisiertes Objekt handelt.

   Wenn Sie nun die BeanBox schliessen und anschliessend neu starten, können Sie das serialisierte Objekt reaktivieren! Besser und sicherer ist es, wenn Sie auch noch die BeanBox speichern und eventuell ein Applet generieren lassen (das kann etwas dauern!).

1.3.3. Lösungshinweise

1.3.3.1. Aufgabe 1 - BDK Starten

Sie können das BDK vom Sun Site herunterladen. Eine Kopie befindet sich (vermutlich die aktuelle Version) auch auf dem Server / der CD.

Der Sun Link wurde bereits mehrfach erwähnt:

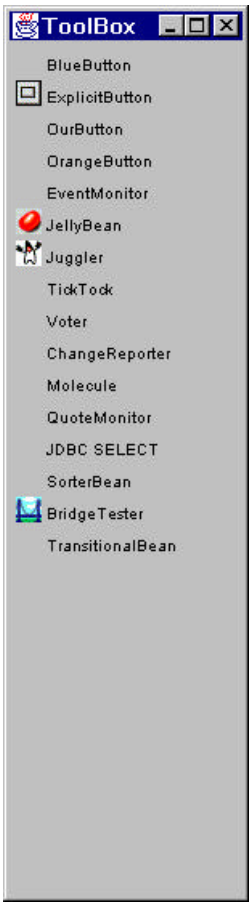
`http://java.sun.com/products/javabeans`

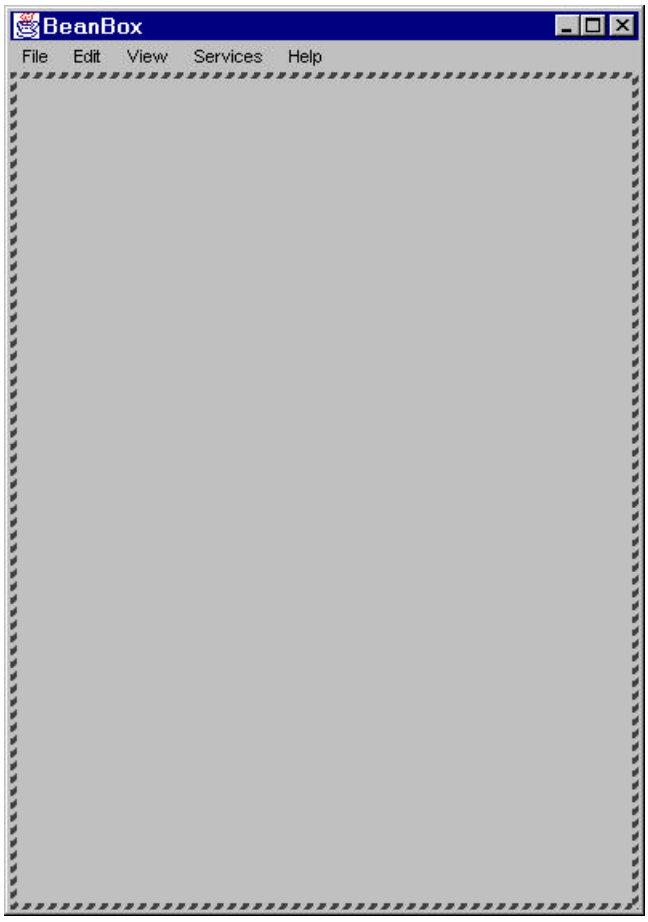
Auch die möglichen Anpassungen der Skripts wurde bereits in der Aufgabenstellung besprochen.

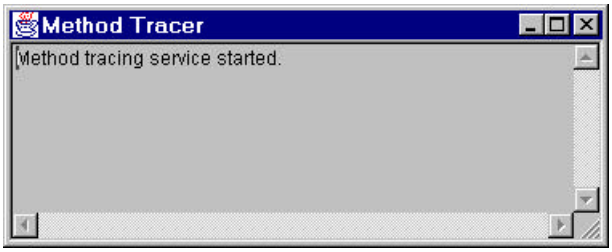
- `cd beanbox`
- `run` [Windows NT/95]
- `run.sh` [UNIX]

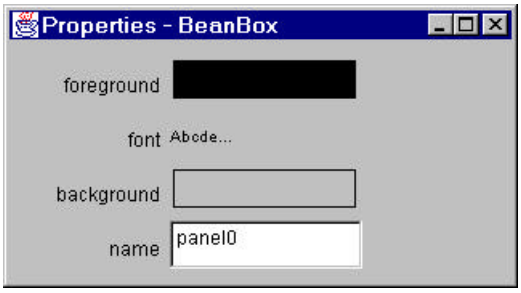
1.3.3.2. Aufgabe 2 - Die Komponenten des BDKs

Hier ein Screen Snapshot der einzelnen BDK Komponenten:



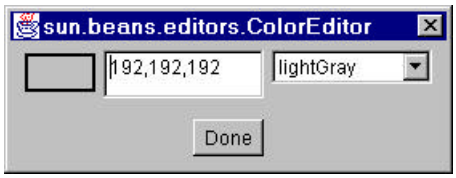






1.3.3.2.1. Editoren des ExplicitButtons

Plazieren Sie den ExplicitButton Bean in die BeanBox und selektieren Sie die Bohne:
Wenn Sie jetzt auf eine der Farben im Property Sheet klicken erscheint der entsprechende Editor



Wenn Sie auf den kleinen Text ganz oben im Property Sheet klicken erscheint der Font Editor.

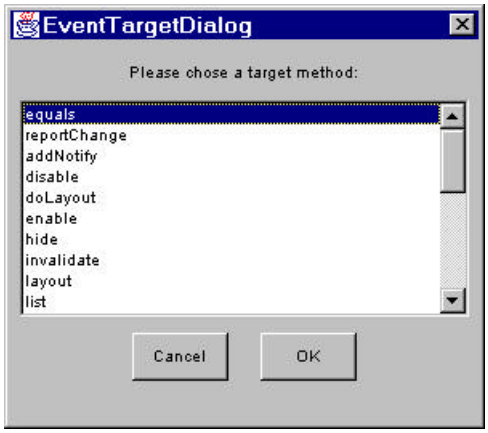
1.3.3.3. Plazieren weiterer Beans in der BeanBox

Wir plazieren beispielsweise das Molekül, JellyBean, SorterBean, ... und spielen etwas damit.
Das BDK hat einige schlechte Eigenschaften:

1) beim Selektieren in der ToolBox sehen Sie nur einfach ein Kreuz, Sie können also nicht genau feststellen, welche Bean gewählt wurde.
2) das Plazieren der Beans ist ungenau wegen fehlendem Raster
3) immer wenn neben der Bean ein Icon erscheint, heisst dies, dass das Bean eine BeanInfo besitzt.

1.3.3.4. Verbinden von Beans

Die Auswahl der Optionen geschieht genau wie oben erklärt. Bei der Verbindung (rote RubberLine) sollten Sie folgendes Menü sehen:

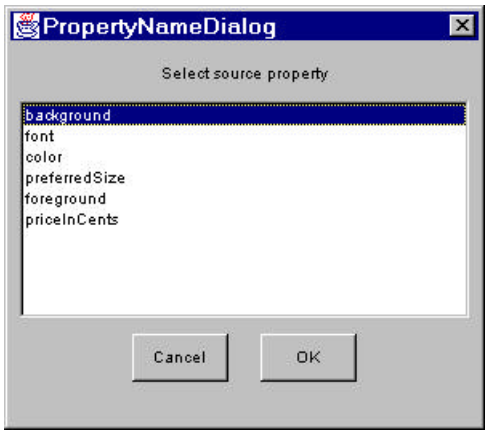


Die zweite Option ('reportChange') wollen wir einsetzen.

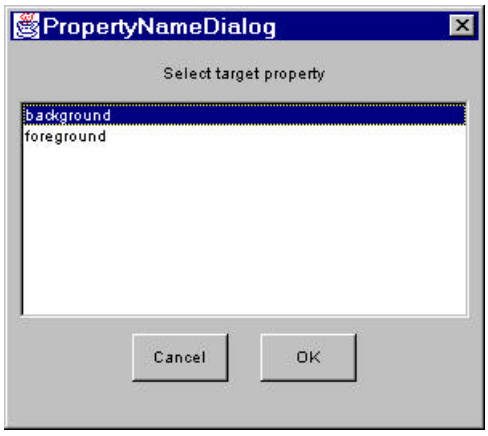
Nach der Auswahl wird javac aufgerufen und der entsprechende Java Code neu übersetzt.

1.3.3.5. Propagieren von Änderungen

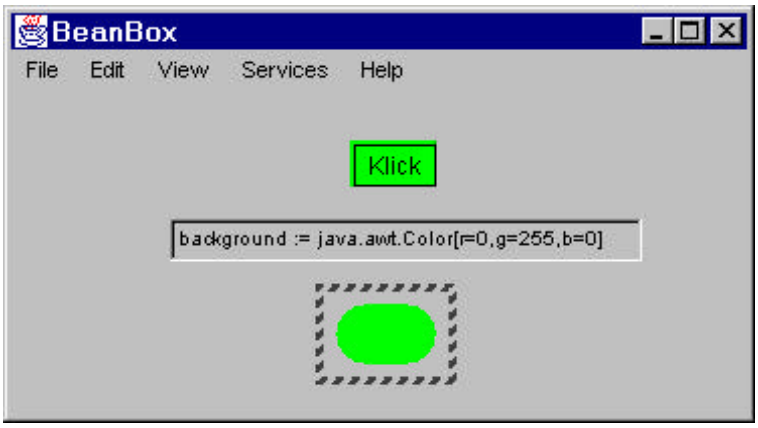
Auch in dieser Aufgabe finden Sie eine detaillierte Anleitung im Aufgabentext.
Das Binden der Properties geschieht im Edit Menü.
Dabei wird folgende Auswahl angezeigt:



Der Button, das Ziel, besitzt weniger Auswahl:

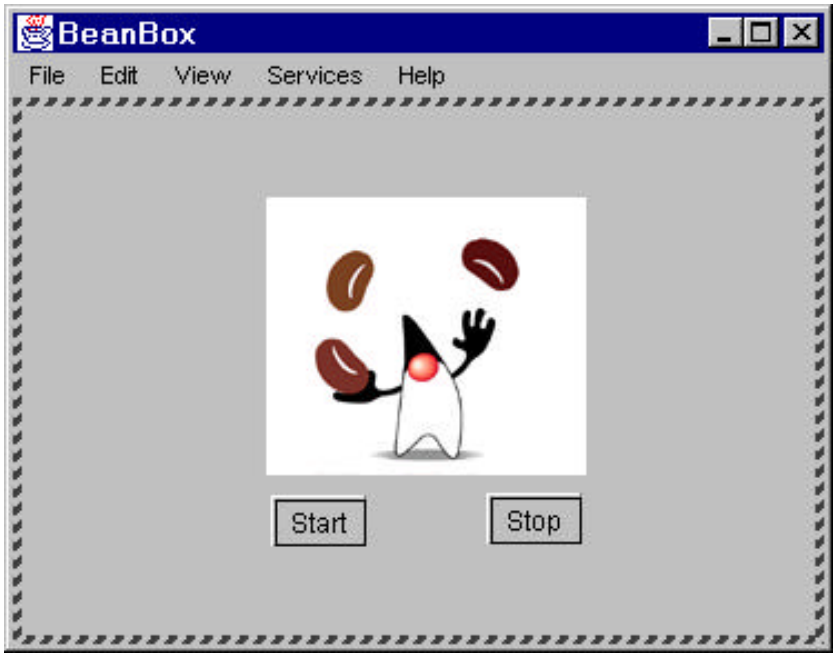


Und das Ergebnis der Verbindung sieht schliesslich folgendermassen aus:
nach dem Ändern der Farbe im JellyButton auf grün, wird die Farbe des ExplicitButtons sofort angepasst:



1.3.3.6. JugglerBean mit Start und Stop

Die Erstellung dieser Bean Kombination ist genau so einfach! Sie müssen wieder die Buttons mit den startjuggling() und stopjuggling() Methoden verbinden und schon sind Sie fertig!



1.3.4. Demos

siehe oben

1.3.5. Musterlösung

Eine Musterlösung auf Java Ebene existiert nicht, weil wir nichts kodiert haben!
Eine Bemerkung zum BDK:
die Serialisierung funktioniert scheinbar nicht 100% zuverlässig. Interessant ist die Code Generierung beim Applet. Sie erkennen, wenn Sie den Quellcode ansehen, dass mit einem serialisierten Objekt (der Bean) gearbeitet wird.

1.4. Kreieren Ihrer ersten Bean

In dieser Übung arbeiten Sie endlich mit dem Java Quellcode einer Bean.

1.4.1. Lernziele

- Sie verstehen eine Bean
- Sie modifizieren eine Bean!

1.4.2. Voraussetzungen

Sie haben das BDK oder ein ähnliches Tool installiert und konfiguriert.

1.4.3. Rumpfprogramme

Die folgenden zwei Rahmenprogramme enthalten im wesentlichen den programmcode, den Sie für Ihre erste Bohne benötigen. Warum? Weil es besser ist, sich zuerst einmal ein bestehendes programm anzusehen, als frustriert das Handtuch zu werfen!

1.4.3.1. KivaHan JavaBean Programmcode

```
import java.awt.*;

public class KivaHan extends Canvas {

  private String msg;
  private int width, height;

  public KivaHan () {
    this ("türkischer Kaffee: 10 lira");//türkische Währung
  }
  public KivaHan (String s) {
    this (s, 200, 200);
  }
  public KivaHan (String s, int width, int height) {
    msg = s;
    this.width = width;
    this.height = height;
    setForeground (SystemColor.controlText);
    setFont (new Font ("Serif", Font.ITALIC, 24));
    setSize (getPreferredSize());
  }
  public void paint (Graphics g) {
    Dimension d = getSize();
    FontMetrics fm = g.getFontMetrics();
    int len = fm.stringWidth (msg);
    int x = Math.max (((d.width - len) / 2), 0);
    int y = d.height / 2;
    g.drawString (msg, x, y);
  }

  public Dimension getPreferredSize () {
    return new Dimension (width, height);
  }
}
```

1.4.3.2. KivaHanTester Java Programmcode

```
public class KivaHanTester extends java.applet.Applet {
  public void init () {
    add (new KivaHan());
  }
}
```

1.4.3.3. Applet HTML Code

Hier die relevante Zeile aus der HTML Seite:

```
<APPLET CODE="KivaHanTester.class" CODEBASE="./Loesung/" HEIGHT="250"
WIDTH="250"></APPLET>
```

1.4.4. Hintergrundinformationen

Gemäss einer Chronologie wurde der Kaffee vom arabischen Ziegenhirt Kaidi um 850 in Ost Afrika entdeckt. Er bemerkte, dass seine Ziegen ausgelassen wurden, nachdem Sie die Beeren eines wilden Busches gekaut hatten.

1475 wurde, wieder gemäss Chrologie, das erste Kaffeehaus der Welt in Constantinopel eröffnet, das Kiva Han.

Soweit zur Allgemeinbildung!

1.4.5. Aufgaben

Diese Aufgabe umfasst fünf einfache Teilaufgaben / Schritte.

Das Rahmenprogramm enthält fix codiert als Text den Kaffeepreis. Ihre Aufgabe besteht darin, diesen Text besser und variabler spezifizieren zu können.

Gehen Sie schrittweise vor und erstellen Sie zuerst eine lauffähige Bean.
1) übersetzen Sie die Dateien
2) schauen Sie sich das Testerprogramm an: es verwendet die definierte Bohne
3) übersetzen Sie das Testerprogramm
4) schauen Sie sich den Applet Rahmen (`KivaHanTester.html`) genauer an
5) testen Sie das Applet mit dem Appletviewer (`appletviewer.exe` im JDK)

1.4.6. Hilfestellungen

Die folgende Hilfestellungen beziehen sich auf die obigen Schritte.

1.4.6.1. Einfachere Handhabung des Textes

Das Muster für die Lösung dieser Aufgabe:

```
public void set<i>PropertyName</i> (<i>PropertyType</i> value);
public <i>PropertyType</i> get<i>PropertyName</i>();
```

Zudem sollten Sie Methoden zur Bestimmung / dem Setzen der Nachricht definieren, beispielsweise

```
get/setMessage()
```

Nach jeder Änderung muss die `repaint()` Methode aufgerufen werden.

1.4.6.2. Übersetzen Sie den Quellcode

Das sollte kein Problem für Sie sein. Auf dem Server / der CD steht Ihnen zudem ein JBuilder Projekt zur Verfügung.

```
javac KivaHan.java
```

1.4.6.3. Das Tester Applet

Überprüfen Sie, ob das Applet die korrekte Bohne verwendet und speichern Sie allenfalls Änderungen ab.

1.4.6.4. Übersetzen des Testers

Das sollte kein Problem für Sie sein. Auf dem Server / der CD steht Ihnen zudem ein JBuilder Projekt zur Verfügung.

```
javac KivaHanTester.java
```

1.4.6.5. Die Tester HTML Seite

Kontrollieren Sie, ob Ihre HTML Seite auf das Tester Applet zeigt (Codebase)

1.4.6.6. Testen Sie das Applet mit der bean im Appletviewer

Der Aufruf des AppletViewers:

```
appletviewer KivaHanTester.html
```

1.4.7. Demos

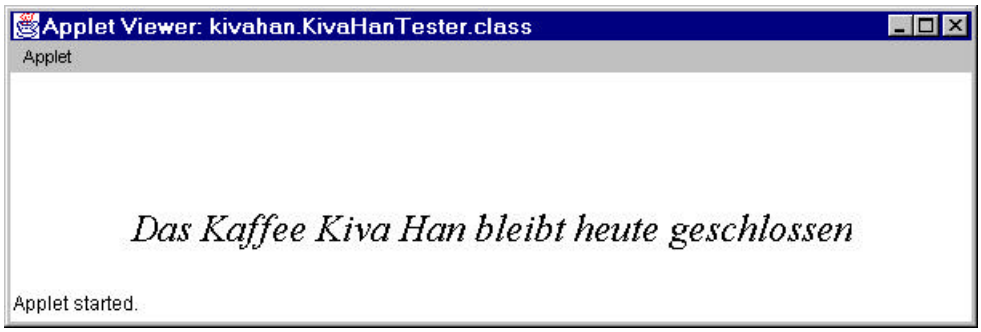
Das Rahmenprogramm liefert folgende Ausgabe:

Der Text kann variieren, je nachdem was Sie eingegeben haben.

1.4.8. Musterlösung

Zu dieser Übung gibt es viele Musterlösungen, da Sie je nach Ihren Vorkenntnissen die eine oder andere Property Variante als Lösung entwickeln werden.

Hier eine mögliche Ausgabe:



mit folgendem Applet:

```
package kivahan;

/**
 * Title:        Kiva Han
 * Description:  EInfache JavaBean mit Textausgabe in einem Applet
 * Copyright:    Copyright (c) J.M.Joller
 * Company:      Joller-Voss
 * @author J.M.Joller
 * @version 1.0
 */

public class KivaHanTester extends java.applet.Applet {
  public void init () {
    String str="Das Kaffee Kiva Han bleibt heute geschlossen";
//    add (new KivaHan(str,str.length()*10+10,100));
    add (new KivaHan(str));
  }
}
```

Dazu wurden die passenden Konstruktoren definiert:

```
package kivahan;

/**
 * Title:        Kiva Han
 * Description:  EInfache JavaBean mit Textausgabe in einem Applet
 * Copyright:    Copyright (c) J.M.Joller
 * Company:      Joller-Voss
 * @author J.M.Joller
 * @version 1.0
 */

import java.awt.*;

public class KivaHan extends Canvas {

  private String msg;
  private int width, height;

  public KivaHan () {
    this ("türkischer Kaffee: 10 lira");
  }
  public KivaHan (String s) {
    this(s, s.length()*10+10, 200); // länge/höhe
  }
  public KivaHan (String s, int width, int height) {
    msg = s;
    this.width = width;
    this.height = height;
    setForeground (SystemColor.controlText);
    setFont (new Font ("Serif", Font.ITALIC, 24));
    setSize (getPreferredSize());
  }
  public void paint (Graphics g) {
    Dimension d = getSize();
    FontMetrics fm = g.getFontMetrics();
    int len = fm.stringWidth (msg);
    int x = Math.max (((d.width - len) / 2), 0);
    int y = d.height / 2;
    g.drawString (msg, x, y);
  }
  public void setMessage (String s) {
    msg = s;
    repaint();
  }
  public String getMessage () {
    return msg;
  }
  public Dimension getPreferredSize () {
    return new Dimension (width, height);
  }
}
```

Noch flexibler wäre die Verwendung von Applet Parametern, so dass der Text in der HTML Seite eingegeben werden könnte.

Wichtig ist, dass Sie sich mit den Mechanismen und der Konstruktion der Bean vertraut gemacht haben. Die Spielereien mit den Parametern sind eher ein Nebeneffekt!

1.5. Erweiterung einer JavaBean

Aufbauend auf der vorherigen Übung bauen Sie die JavaBean aus. An Stelle einer fixen Text Area verwenden Sie dazu rollbare Flächen, Events und constrained Properties.

Als Basis erhalten Sie ein Rahmenprogramm, welches Sie zu einer Bean machen müssen.

1.5.1. Lernziele

- Sie lernen, wie man Bean Komponenten kreiert
- Sie arbeiten mit constrained Properties
- Sie definieren und arbeiten mit Source Events
- Sie stellen erste Überlegungen an, um Persistenz einzubauen

1.5.2. Voraussetzungen

Sie haben die vorhergehenden Übungen durchgearbeitet.

1.5.3. Rumpfprogramme

1.5.3.1. CentralPerkTester

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import java.beans.*;

class PropLabel extends Label implements PropertyChangeListener {
  public void propertyChange(PropertyChangeEvent e) {
    if (e.getPropertyName().equals("message")) {
      setText((String)e.getNewValue());
    }
  };
}

public class CentralPerkTester extends Applet implements PerkListener {
  public void init () {
    setLayout (new BorderLayout());
    final CentralPerk cp = new CentralPerk();
    add (cp, BorderLayout.CENTER);
    PropLabel l = new PropLabel();
    add (l, BorderLayout.NORTH);
    cp.addPropertyChangeListener (l);
    cp.addPerkListener (this);
    Panel p = new Panel (new GridLayout(0, 2));
    Label tl = new Label ("Bitte neuen Text eingeben:");
    p.add (tl);
    TextField tf1 = new TextField("", 20);
    p.add (tf1);
    tf1.addActionListener ( new ActionListener() {
      public void actionPerformed (ActionEvent e) {
        cp.setMessage (e.getActionCommand());
      }
    });
    Label t2 = new Label ("Neue Erneuerungsrate eingeben:");
    p.add (t2);
    TextField tf2 = new TextField();
    p.add (tf2);
    tf2.addActionListener ( new ActionListener() {
```

```
      public void actionPerformed (ActionEvent e) {
        try {
          cp.setMovingRate (Integer.parseInt(e.getActionCommand()));
        } catch (NumberFormatException ne) {
          cp.setMovingRate (cp.getMovingRate());
        }
      }
    });
    add (p, BorderLayout.SOUTH);
  }
  public void startedPerking (PerkEvent e) {
    System.out.println ("Message: " + e.getMessage());
  }
}
```

1.5.3.2. CentralPerk

```
import java.awt.*;
import java.io.*;

public class CentralPerk extends Canvas implements Runnable {

  private String msg;
  private int x, y, width, height;
  private int rate = 100;
  private Thread mover;
  private boolean stopped = false;

  public CentralPerk () {
    this ("Coffee: 10 kurus");
  }
  public CentralPerk (String s) {
    this (s, 200, 200);
  }
  public CentralPerk (String s, int width, int height) {
    msg = s;
    this.width = width;
    this.height = height;
    setSize (getPreferredSize());
    initMoving();
  }
  private void initMoving() {
    mover = new Thread(this);
    mover.start();
  }
  public Dimension getPreferredSize () {
    return new Dimension (width, height);
  }
  public void paint (Graphics g) {
    Dimension d = getSize();
    if (x <= 0) {
      // Restart scrolling
      x = d.width;
      y = d.height / 2;
    } else {
      x--;
    }
    g.drawString (msg, x, y);
  }

// Properties
```

```
  public void setMessage(String message) {
    msg = message;
    repaint();
  }
  public String getMessage () {
    return msg;
  }

// Animation

  public void start () {
    startMoving();
  }
  public void stop () {
    stopMoving();
  }
  public void run () {
    try {
      while(true) {
        // warten.
        synchronized (this) {
          while (stopped || !isEnabled()) {
            wait();
          }
        }
        // Frame zeichnen.
        repaint();
        mover.sleep(rate);
      }
    } catch (InterruptedException e) {
      // Ignorieren
    }
  }
  public synchronized void setEnabled(boolean x) {
    super.setEnabled(x);
    notify();
  }
  public synchronized void startMoving() {
    stopped = false;
    notify();
  }
  public synchronized void stopMoving() {
    stopped = true;
  }

// IHRE AUFGABE
// Property Changing


// Event Listeners


// Serialization

}
```

1.5.3.3. PerkEvent

```
public class PerkEvent {

}
```

1.5.3.4. PerkListener

```
public interface PerkListener {

}
```

1.5.4. Aufgaben

Diese Übung baut auf der vorhergehenden auf und baut die JavaBean aus. Diese Bean verfügt über ein besseres Texthandling, Events und Listeners und Constrained Properties.

Die obigen Rahmen zeigen, wie die Bean aussehen könnte. Ihre Aufgabe ist es, daraus eine JavaBean zu bauen.

Die Aufgaben im Einzelnen:

1) Die neue JavaBean besitzt zwei Parameter, die Message / den Text und die Erneuerungsrate, grob die Zeit, die benötigt wird, um den Text über das Canvas zu bewegen. Andere Parameter, wie Font, Farbe usw. werden über die Canvas bestimmt. In dem Rahmenprogramm finden Sie get/set Methoden für den Text. Ergänzen Sie das Programm durch Methoden für die Erneuerungsrate.

2) Damit unsere Eigenschaften gebunden sind, benötigen wir Listener und eine `PropertyChangeSupport` Liste. Zudem benötigen wir `set` Methoden, um die geänderten Eigenschaften bekanntzugeben, beispielsweise `setMessage()` und `setMovingRate()`.

3) Immer wenn die Meldung von rechts her sichtbar wird, ins Bild geschoben wird, soll ein `PerkEvent` generiert werden. Dieses muss definiert werden und soll als read-only Message Property definiert sein (also lediglich über eine get Methode.verfügen).

4) Als nächstes benötigen wir eine Listener, `PerkListener`, mit einer Methode, mit der wir benachricht werden können, wenn das `PerkEvent` eintrifft. Diese Merhode nennen wir `startedPerking()`.

5) In der Klasse CentralPerk müssen wir die PerkListener Liste unterhalten und Listener hinzufügen und entfernen. Zudem müssen die Listener über ihre Ereignisse informiert werden.
6) Als letzte Aufgabe müssen wir dafür sorgen, dass die Bohne serialisierbar ist. Dabei überschreiben wir die Standard-Serialisierung so, dass beim Lesen (`readObject()`) die transienten Variablen initialisiert werden.

1.5.5. Hilfestellungen

1) Die Erneuerungsrate benötigt Methoden mit folgender Signatur:
```
public void setMovingRate(int rate)
public int getMovingRate()
```

2) Die Listener definieren wir als privat:
```
private PropertyChangeSupport changes = new PropertyChangeSupport
(this);
```

   Die Listener add/remove Methoden müssen public sein:

```
public void addPropertyChangeListener (PropertyChangeListener p) {
   changes.addPropertyChangeListener (p);
}
public void removePropertyChangeListener (
   PropertyChangeListener p) {
   changes.removePropertyChangeListener (p);
}
```

3) Das Bean Event muss eine Unterklasse von `java.util.EventObject` sein.
   Zudem muss eine `getMessage()` Methode kreiert werden, um die Message Properties zu bestimmen.

4) Die Listener müssen `java.util.EventListener` implementieren.

5) Die Verwaltung der EventListener geschieht wie üblich mit einem Vector:
   - die `addListener` Methode wird damit zu `addElement()`
   - die `removeListener` Methode entspricht entspricht der `removeElement()` Methode

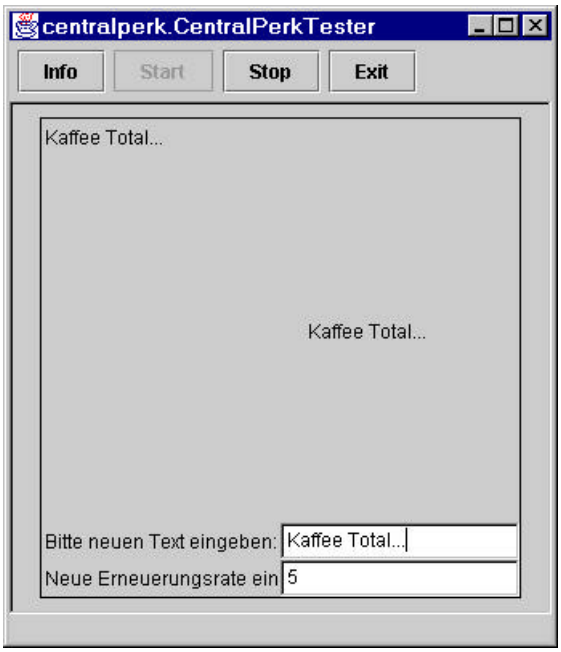
6) Die Basisdatentypen sind alle serialisierbar. Damit haben wir keine grösseren Probleme mit der Serailisierung. Threads sind dies aber nicht. Threads müssen als transient gekennzeichnet werden.

7) Das Überschreiben der Serialisierung ist nicht besonders schwierig:
   hier die Signaturen der Methoden
```
private void writeObject(ObjectOutputStream s)
throws IOException

private void readObject(ObjectInputStream s)
throws ClassNotFoundException, IOException
```

8) Jetzt können Sie die Programme testen.

1.5.6. Demo



1.5.7. Musterkösung

Auch zu dieser Aufgabe gibt es viele mögliche Lösungen. Im folgenden sehen Sie eine der möglichen:

1.5.7.1. CentralPerkTester

```
package centralperk;

/**
 * Title:        Edles Kaffeehaus in den Staaten
 * Description:  Weiterentwicklung der Kaidi JavaBean zu einer
funktionsreicheren Variante
 * Copyright:    Copyright (c) J.M.Joller
 * Company:      Joller-Voss
 * @author J.M.Joller
 * @version 1.0
 */

import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import java.beans.*;
import javax.swing.*;

class PropLabel extends Label implements PropertyChangeListener {
  public void propertyChange(PropertyChangeEvent e) {
    if (e.getPropertyName().equals("message")) {
      setText((String)e.getNewValue());
    }
  };
}

public class CentralPerkTester extends Applet implements PerkListener {
```

```
  public void init () {
    setLayout (new BorderLayout());
    final CentralPerk cp = new CentralPerk();
    add (cp, BorderLayout.CENTER);
    PropLabel l = new PropLabel();
    add (l, BorderLayout.NORTH);
    cp.addPropertyChangeListener (l);
    cp.addPerkListener (this);
    Panel p = new Panel (new GridLayout(0, 2));
    Label tl = new Label ("Bitte neuen Text eingeben:");
    p.add (tl);
    TextField tf1 = new TextField("", 20);
    p.add (tf1);
    tf1.addActionListener ( new ActionListener() {
      public void actionPerformed (ActionEvent e) {
        cp.setMessage (e.getActionCommand());
      }
    });
    Label t2 = new Label ("Neue Erneuerungsrate eingeben:");
    p.add (t2);
    TextField tf2 = new TextField();
    p.add (tf2);
    tf2.addActionListener ( new ActionListener() {
      public void actionPerformed (ActionEvent e) {
        try {
          cp.setMovingRate (Integer.parseInt(e.getActionCommand()));
        } catch (NumberFormatException ne) {
          cp.setMovingRate (cp.getMovingRate());
        }
      }
    });
    add (p, BorderLayout.SOUTH);
  }
  public void startedPerking (PerkEvent e) {
    System.out.println ("Message: " + e.getMessage());
  }
}
```

1.5.7.2. CentralPerk

```
package centralperk;

/**
 * Title:        Edles Kaffeehaus in den Staaten
 * Description:  Weiterentwicklung der Kaidi JavaBean zu einer
funktionsreicheren Variante
 * Copyright:    Copyright (c) J.M.Joller
 * Company:      Joller-Voss
 * @author J.M.Joller
 * @version 1.0
 */

import java.awt.*;
import java.beans.*;
import java.io.*;
import java.util.*;

public class CentralPerk extends Canvas implements Runnable {

  private String msg;
  private int x, y, width, height;
```

```
  private int rate = 100;
  private transient Thread mover;
  private boolean stopped = false;

  public CentralPerk () {
    this ("Coffee: 10 kurus");
  }
  public CentralPerk (String s) {
    this (s, 200, 200);
  }
  public CentralPerk (String s, int width, int height) {
    msg = s;
    this.width = width;
    this.height = height;
    setSize (getPreferredSize());
    initMoving();
  }
  private void initMoving() {
    mover = new Thread(this);
    mover.start();
  }
  public Dimension getPreferredSize () {
    return new Dimension (width, height);
  }
  public void paint (Graphics g) {
    Dimension d = getSize();
    if (x <= 0) {
      x = d.width;
      y = d.height / 2;
      notifyPerked();
    } else {
      x--;
    }
    g.drawString (msg, x, y);
  }

// Properties

  public void setMessage(String message) {
    if (msg != message) {
      String oldMessage = msg;
      msg = message;
      repaint();
      changes.firePropertyChange (
        "message", oldMessage, msg);
    }
  }
  public String getMessage () {
    return msg;
  }
  public void setMovingRate (int rate) {
    if (this.rate != rate) {
      Integer oldRate = new Integer (this.rate);
      this.rate = rate;
      changes.firePropertyChange (
        "rate", oldRate, new Integer (this.rate));
    }
  }
  public int getMovingRate () {
    return rate;
  }
```

```
// Animation

  public void start () {
    startMoving();
  }
  public void stop () {
    stopMoving();
  }
  public void run () {
    try {
      while(true) {
        // warten, bis die Animation gestoppt ist
        synchronized (this) {
          while (stopped || !isEnabled()) {
            wait();
          }
        }
        // aktuelles Frame anzeigen
        repaint();
        mover.sleep(rate);
      }
    } catch (InterruptedException e) {
      // ignorieren
    }
  }
  public synchronized void setEnabled(boolean x) {
    super.setEnabled(x);
    notify();
  }
  public synchronized void startMoving() {
    stopped = false;
    notify();
  }
  public synchronized void stopMoving() {
    stopped = true;
  }

// Properties

  private PropertyChangeSupport changes =
    new PropertyChangeSupport (this);

  public void addPropertyChangeListener (
      PropertyChangeListener p) {
    changes.addPropertyChangeListener (p);
  }
  public void removePropertyChangeListener (
      PropertyChangeListener p) {
    changes.removePropertyChangeListener (p);
  }

// Event Listeners

  private Vector perkListeners = new Vector();
  public synchronized void addPerkListener (PerkListener l) {
    perkListeners.addElement (l);
  }
  public synchronized void removePerkListener (PerkListener l) {
    perkListeners.removeElement (l);
  }

  protected void notifyPerked () {
```

```
    Vector l;
    // Create Event
    PerkEvent p = new PerkEvent (this, msg);
    // Sicherheitskopie
    synchronized (this) {
      l = (Vector)perkListeners.clone();
    }
    for (int i=0;i<l.size();i++) {
      PerkListener pl = (PerkListener)l.elementAt (i);
      pl.startedPerking(p);
    }
  }

// Serialization

  private void writeObject(ObjectOutputStream s)
      throws IOException {
    s.defaultWriteObject();
  }
  private void readObject(ObjectInputStream s)
      throws ClassNotFoundException, IOException {
    s.defaultReadObject();
    initMoving();
  }
}
```

1.5.7.3. PerkEvent

```
package centralperk;

import java.util.EventObject;
public class PerkEvent extends EventObject {
  private String msg;
  public PerkEvent (Object source) {
    this (source, null);
  }
  public PerkEvent (Object source, String message) {
    super (source);
    msg = message;
  }
  public String getMessage () {
    return msg;
  }
}
```

1.5.7.4. PerkListener

```
package centralperk;

public interface PerkListener
    extends java.util.EventListener {
  public abstract void startedPerking (PerkEvent e);
}
```

1.6. Inspektion von Beans mit BeanInfo

Mit Hilfe von Reflection und BeanInfo bauen Sie Zusatzinformationen in Ihre JavaBeans ein. Zusätzlich verwenden Sie das Reflection API. Sie verwenden zusätzlich Methoden, welche es Ihnen gestatten, die Informationen zu bestimmen und auszuwerten (Properties, Event Listener).

Ihre Aufgabe besteht darin, die Inspektionsmethode des Rahmenprogramms zu ergänzen, zu vervollständigen, um alle Namen und Werte der Properties für alle Event Listener der Bohne zu bestimmen.

1.6.1. Lernziele

- Sie lernen, wie man eine Bohne inspiziert, mit Hilfe der BeanInfo
- Sie bestimmen Events und Properties einer Bohne
- Sie rufen Methoden auf, ohne deren Namen zu kennen.

1.6.2. Voraussetzungen

Sie haben das BDK installiert und die vorangehenden Übungen durchgearbeitet.

1.6.3. Rahmenprogramme

Die folgenden Rahmenprogramme dienen Ihnen als Vorlage zur Lösung der Aufgaben.

1.6.3.1. Valdez

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

public class Valdez {
  public static void main (String args[]) {
    TextField tf = new TextField ("Hallo");
    inspect (tf);
  }

  public static void inspect (Object o) {
    inspect (o, System.out);
  }
  public static void inspect (Object o, OutputStream os) {
    inspect (o, new PrintWriter (os));
  }
  public static void inspect (Object o, Writer w) {
    inspect (o, new PrintWriter (w));
  }
  public static void inspect (Object o, PrintWriter pw) {
    try {
      // Bestimme die Klasse des Objekts o

      // Lies die BeanInfo Klasse

      // Lies die PropertyDescriptor Liste

      // Lies die EventSetDescriptor Liste
```

```
      pw.println ("Bean Information für: " + c);

      Method readMethod;
      for (int i=0;i<pd.length;i++) {
        if (i==0) pw.println ("\n\tProperties\n\t----------");
        pw.print ("(" + (i+1) + ")\t");
        // Ausgabe von Name und Property Type
        pw.print ( /* IHRE AUFGABE: Name and Type */);

        // Lese Methoden der Property
        readMethod = /* IHRE AUFGABE: Lese Methoden */;

        if (readMethod != null) {
          try {
            // Aufruf der read Methode und Ausgabe

            pw.println (" - " + /* IHRE AUFGABE : Ausgabe */ );

          } catch (IllegalArgumentException e) {
            pw.println ("Inspektionsfehler - Illegales Argument");
          } catch (IllegalAccessException e) {
            pw.println ("Inspektionsfehler - Illegaler Zugriff");
          } catch (InvocationTargetException e) {
            pw.println ("Inspektionsfehler - Invocation Probleme");
          }
        } else {
          pw.println ();
        }
      }
      for (int i=0;i<esd.length;i++) {
        if (i==0) pw.println ("\n\tListeners\n\t---------");
        pw.print ("(" + (i+1) + ")\t");
        // Ausgabe der Namen und Listener Typen
        pw.print ( /* IHRE AUFGABE: Name und Type */);

      }
    } catch (IntrospectionException e) {
      pw.println ("Introspection Fehler");
    }
    pw.flush();
  }
}
```

1.6.3.2. Burro

```
import java.awt.*;
import java.awt.event.*;
import java.io.*;
public class Burro extends Frame implements ActionListener, ItemListener {

  TextArea ta;

  public Burro () {
    super ("Burro");
    ta = new TextArea ();
    ta.setEditable(false);
    add (ta, BorderLayout.CENTER);
    CheckboxGroup cb = new CheckboxGroup ();
    Panel p = new Panel(new GridLayout (0, 1));
    String fonts[] = getToolkit().getFontList();
    Checkbox c;
```

```
    for (int i=0;i<fonts.length;i++) {
      p.add (c = new Checkbox (fonts[i], false, cb));
      c.addItemListener (this);
    }
    add (p, BorderLayout.EAST);
    TextField tf = new TextField ();

    tf.setEchoChar ('*');
    tf.addActionListener (this);
    add (tf, BorderLayout.SOUTH);
    List l = new List();
    for (int i=0;i<fonts.length;i++) {
      l.add (fonts[i]);
    }
    l.addItemListener (this);
    add (l, BorderLayout.WEST);
    add (new Label (
      "Selektieren Sie irgend etwas und lesen Sie die BeanInfo in der Text
Area"),
      BorderLayout.NORTH);
    enableEvents (AWTEvent.WINDOW_EVENT_MASK);
    pack ();
    show();
  }

  public static void main (String args[]) {
    Burro b = new Burro ();
  }

  public void itemStateChanged (ItemEvent e) {
    inspectIt (e);
  }

  public void actionPerformed (ActionEvent e) {
    inspectIt (e);
  }

  protected void processWindowEvent(WindowEvent e) {
    super.processWindowEvent(e);
    if (e.getID() == WindowEvent.WINDOW_CLOSING)
      System.exit(0);
  }

  public void inspectIt (AWTEvent e) {
    StringWriter sw = new StringWriter();
    Valdez.inspect (e.getSource(), sw);
    ta.setText(sw.toString());
  }
}
```

1.6.4. Aufgaben

Mit Hilfe der BeanInfo und dem Reflection API soll die vorgegebene Bean inspiziert werden. Diese Übung zeigt Ihnen, wie Sie mit BeanInfo die Eigenschaften einer JavaBean abfragen können und die EventListener bestimmen können.

Ihre Aufgabe besteht darin, die inspect() Methode des Rahmenprogramms zu vervollständigen, um alle Namen und Werte der Properties und Listener der Bean auszugeben.

Die Aufgaben im Einzelnen:

1) Die `Inspector` Klasse wird eingesetzt, um die `BeanInfo` einer Bean zu erhalten. Da diese eine Instanz der Klasse `Class` benötigt, müssen Sie die Class der Bean bestimmen und der `inspect()` Methode mitgeben.

2) Mit der `Class` können Sie mit Hilfe des `Inspector` die BeanInfo bestimmen.

3) Aus `BeanInfo` können Sie die Liste der Eigenschaften der Bean erhalten: als Array von `PropertyDescriptor` Objekten.

4) Geben Sie für jedes Element des `PropertyDescriptor` Arrays den Namen und den aktuellen Wert aus.

5) Bestimmen Sie jetzt das Event Set für die Bohne. Dies liefert Ihnen ein Array von `EventSetDescriptor` Objekten.

6) Geben Sie für jedes Element des Event Sets den Namen und den Listener Typus aus.

7) Prüfen Sie mit Hilfe des Burro Frames die inspect() Methode, Burro kreiert ein Frame mit Beans. Immer wenn Sie Änderungen vornehmen (Text, ...) werden diese in der Textfläche angezeigt.

1.6.5. Hilfestellung

Zu jeder Aufgabe existiert ein kurzer Lösungshinweis. Weitere Informationen erkennen Sie aus der Musterausgabe / der Demo und der Musterlösung.

1) Zum Bestimmen der Class verwenden Sie die `Object.getClass()` Methode.

2) Instrospector verfügt über eine Methode `getBeanInfo()`. Sie können also `Introspection.getBeanInfo()` verwenden. Beachten Sie, dass eine `IntrospectionException` geworfen werden kann.

3) Das Array erhalten Sie durch Aufruf der Methode `BeanInfo.getPropertiyDescriptors()`

4) Die `PropertyDescriptor` Klasse ist eine Unterklasse von `FeatureDescriptor`. Mit der Methode `FeatureDescriptor.getName()` erhalten Sie den Namen; mit `PropertyDescriptor.getPropertyType()` erhalten Sie die `Class` für jede Property;
   mit `PropertyDescriptor.getReadMethod()` erhalten Sie die `read()` Methode der Property;
   mit `Method.invokeMethod()` können Sie diese ausführen.

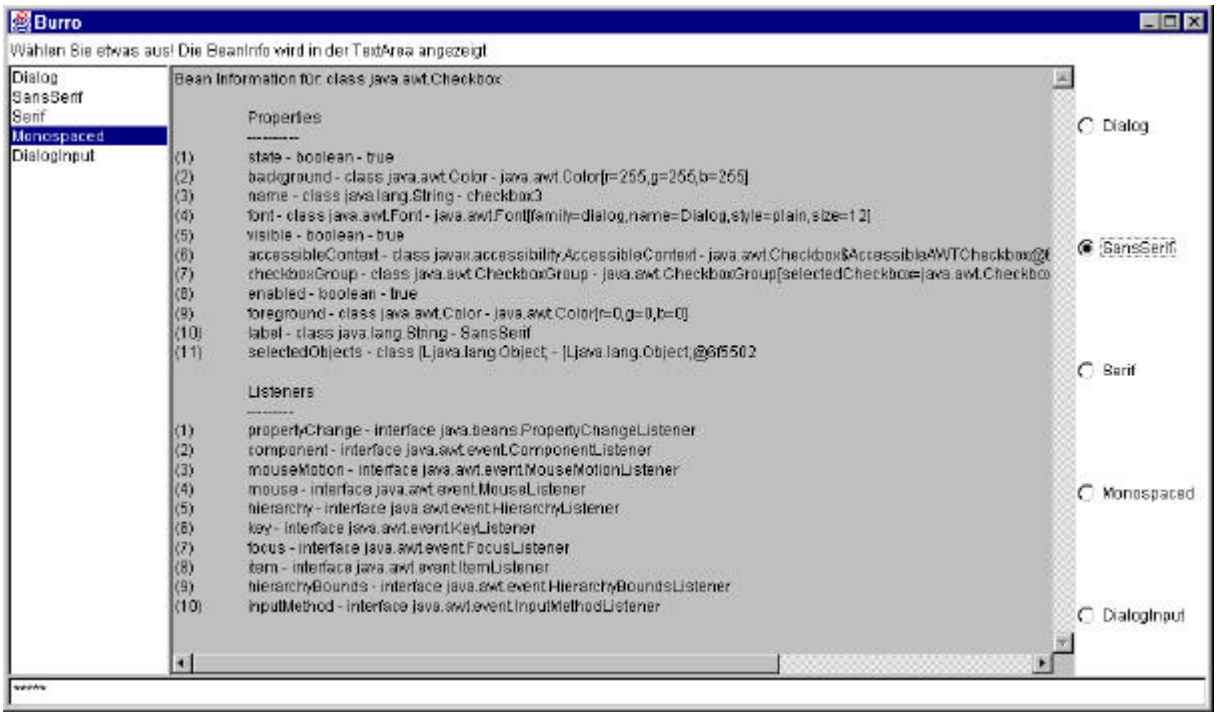
   Folgende Ausnahmen können von der `invoke()` Methode geworfen werden: `IllegalArgumentException`, `IllegalAccessException` oder `InvocationTargetException`.

5) Die Klasse `BeanInfo` liefert über die Methode `getEventSetDescriptor()` ein Array von `EventSetDescriptor` Objekten.

6) Auch hier verwenden wir die Klassenhierarchie:
   `EventSetDescriptor` ist eine Unterklasse von `FeatureDescriptor`.
   `FeatureDescriptor` besitzt eine Methode `getName()`. welche den Namen liefert.
   Analog wie oben können wir mit der `getListenerType()` Methode der `EventDescriptor` Klasse den Typus, die `Class` herausfinden.

7) Jetzt müssen Sie nur noch alles übersetzen und `Burro` starten.

1.6.6. Demo



1.6.7. Musterlösung

1.6.7.1. Burro

```
package beaninfobeispiel;

/**
 * Title:        BeanInfoBeispiel
 * Description:  Einfaches Beispiel zur Illustration der BeanInfo
 * Copyright:    Copyright (c) J.M.Joller
 * Company:      Joller-Voss
 * @author J.M.Joller
 * @version 1.0
 */

import java.awt.*;
import java.awt.event.*;
import java.io.*;
public class Burro extends Frame implements ActionListener, ItemListener {

  TextArea ta;

  public Burro () {
    super ("Burro");
    ta = new TextArea ();
    ta.setEditable(false);
    add (ta, BorderLayout.CENTER);
    CheckboxGroup cb = new CheckboxGroup ();
    Panel p = new Panel(new GridLayout (0, 1));
    String fonts[] = getToolkit().getFontList();
```

```
    Checkbox c;
    for (int i=0;i<fonts.length;i++) {
      p.add (c = new Checkbox (fonts[i], false, cb));
      c.addItemListener (this);
    }
    add (p, BorderLayout.EAST);
    TextField tf = new TextField ();
    // Eliminieren eines Fehler:
    // Ohne Echo wird ein Fehler
    // in Valdez.inspect() generiert.
    tf.setEchoChar ('*');
    tf.addActionListener (this);
    add (tf, BorderLayout.SOUTH);
    List l = new List();
    for (int i=0;i<fonts.length;i++) {
      l.add (fonts[i]);
    }
    l.addItemListener (this);
    add (l, BorderLayout.WEST);
    add (new Label (
      "Wählen Sie etwas aus! Die BeanInfo wird in der TextArea angezeigt"),
      BorderLayout.NORTH);
    enableEvents (AWTEvent.WINDOW_EVENT_MASK);
    pack ();
    show();
  }

  public static void main (String args[]) {
    Burro b = new Burro ();
  }

  public void itemStateChanged (ItemEvent e) {
    inspectIt (e);
  }

  public void actionPerformed (ActionEvent e) {
    inspectIt (e);
  }

  protected void processWindowEvent(WindowEvent e) {
    super.processWindowEvent(e);
    if (e.getID() == WindowEvent.WINDOW_CLOSING)
      System.exit(0);
  }

  public void inspectIt (AWTEvent e) {
    StringWriter sw = new StringWriter();
    Valdez.inspect (e.getSource(), sw);
    ta.setText(sw.toString());
  }
}
```

1.6.7.2. Valdez

```
package beaninfobeispiel;
import java.awt.*;
import java.beans.*;
import java.io.*;
import java.lang.reflect.*;
public class Valdez {
  public static void main (String args[]) {
    TextField tf = new TextField ("Hallo");
    inspect (tf);
  }
  public static void inspect (Object o) {
    inspect (o, System.out);
  }
  public static void inspect (Object o, OutputStream os) {
    inspect (o, new PrintWriter (os));
  }
  public static void inspect (Object o, Writer w) {
    inspect (o, new PrintWriter (w));
  }
  public static void inspect (Object o, PrintWriter pw) {
    try {
      Class c = o.getClass();
      BeanInfo bi = Introspector.getBeanInfo (c);
      EventSetDescriptor esd[] = bi.getEventSetDescriptors();
      PropertyDescriptor pd[] = bi.getPropertyDescriptors();
      Method readMethod;
      pw.println ("Bean Information für: " + c);
      for (int i=0;i<pd.length;i++) {
        if (i==0) pw.println ("\n\tProperties\n\t----------");
        pw.print ("(" + (i+1) + ")\t");
        pw.print (pd[i].getName() + " - " + pd[i].getPropertyType());
        readMethod = pd[i].getReadMethod();
        if (readMethod != null) {
          try {
            pw.println (" - " + readMethod.invoke (o, (Object[])null));
          } catch (IllegalArgumentException e) {
            pw.println ("Inspektion schlug fehl - Illegales Argument");
          } catch (IllegalAccessException e) {
            pw.println ("Inspektion schlug fehl - Illegaler Zugriff");
          } catch (InvocationTargetException e) {
            pw.println ("Inspektion schlugfehl - Probleme beim Aufruf der
invoke() Methode");
          }
        } else { pw.println (); }
      }
      for (int i=0;i<esd.length;i++) {
        if (i==0) pw.println ("\n\tListeners\n\t---------");
        pw.print ("(" + (i+1) + ")\t");
        pw.println (esd[i].getName() + " - " + esd[i].getListenerType());
      }
    } catch (IntrospectionException e) {
      pw.println ("Introspection Fehler");
    }
    pw.flush();
  }
}
```

1.7. Zusammenfassen mehrerer Beans

In dieser Übung packen Sie mehrere Beans zusammen und bilden ein Java Archiv, welches auch mit der BeanBox verwendet werden kann. Zusätzlich schreiben Sie eine BeanInfo Klasse.

1.7.1. Lernziele

- Sie lernen Beans zu verpacken
- Sie lernen, wie man eine Manifest Datei kreiert
- Sie lernen, wie eine Bohne in die BeanBox / ToolBox geladen wird.

1.7.2. Voraussetzungen

Sie haben die vorangehende Übung durchgearbeitet.

1.7.3. Rahmenprogramme

1.7.3.1. CentralPerk

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

public class CentralPerk extends Canvas implements Runnable {

  private String msg;
  private int x, y, width, height;
  private int rate = 100;
  private transient Thread mover;
  private boolean stopped = false;

  public CentralPerk () {
    this ("Coffee: 10 kurus");
  }
  public CentralPerk (String s) {
    this (s, 200, 200);
  }
  public CentralPerk (String s, int width, int height) {
    msg = s;
    this.width = width;
    this.height = height;
    setSize (getPreferredSize());
    initMoving();
  }
  private void initMoving() {
    mover = new Thread(this);
    mover.start();
  }
  public Dimension getPreferredSize () {
    return new Dimension (width, height);
  }
  public void paint (Graphics g) {
    Dimension d = getSize();
    if (x <= 0) {
      x = d.width;
      y = d.height / 2;
      notifyPerked();
    } else {
```

```
      x--;
    }
    g.drawString (msg, x, y);
  }

// Properties

  public void setMessage(String message) {
    if (msg != message) {
      String oldMessage = msg;
      msg = message;
      repaint();
      changes.firePropertyChange (
        "message", oldMessage, msg);
    }
  }
  public String getMessage () {
    return msg;
  }
  public void setMovingRate (int rate) {
    if (this.rate != rate) {
      Integer oldRate = new Integer (this.rate);
      this.rate = rate;
      changes.firePropertyChange (
        "rate", oldRate, new Integer (this.rate));
    }
  }
  public int getMovingRate () {
    return rate;
  }

// Animation

  public void start () {
    startMoving();
  }
  public void stop () {
    stopMoving();
  }
  public void run () {
    try {
      while(true) {
        // warten.
        synchronized (this) {
          while (stopped || !isEnabled()) {
            wait();
          }
        }
        // aktuelles Frame anzeigen
        repaint();
        mover.sleep(rate);
      }
    } catch (InterruptedException e) {
      // Ignore
    }
  }
  public synchronized void setEnabled(boolean x) {
    super.setEnabled(x);
    notify();
  }
  public synchronized void startMoving() {
    stopped = false;
```

```
    notify();
  }
  public synchronized void stopMoving() {
    stopped = true;
  }

// Property Änderungen

  private PropertyChangeSupport changes =
    new PropertyChangeSupport (this);

  public void addPropertyChangeListener (
      PropertyChangeListener p) {
    changes.addPropertyChangeListener (p);
  }
  public void removePropertyChangeListener (
      PropertyChangeListener p) {
    changes.removePropertyChangeListener (p);
  }

// Event Listeners

  private Vector perkListeners = new Vector();
  public synchronized void addPerkListener (PerkListener l) {
    perkListeners.addElement (l);
  }
  public synchronized void removePerkListener (PerkListener l) {
    perkListeners.removeElement (l);
  }

  protected void notifyPerked () {
    Vector l;
    // Event kreieren
    PerkEvent p = new PerkEvent (this, msg);
    // Sicherheitskopie erstellen
    synchronized (this) {
      l = (Vector)perkListeners.clone();
    }
    for (int i=0;i<l.size();i++) {
      PerkListener pl = (PerkListener)l.elementAt (i);
      pl.startedPerking(p);
    }
  }

// Serialization

  private void writeObject(ObjectOutputStream s)
      throws IOException {
    s.defaultWriteObject();
  }
  private void readObject(ObjectInputStream s)
      throws ClassNotFoundException, IOException {
    s.defaultReadObject();
    initMoving();
  }
}
```

1.7.4. PerkEvent

```
import java.util.EventObject;
public class PerkEvent extends EventObject {
  private String msg;
  public PerkEvent (Object source) {
    this (source, null);
  }
  public PerkEvent (Object source, String message) {
    super (source);
    msg = message;
  }
  public String getMessage () {
    return msg;
  }
}
```

1.7.5. PerkListener

```
public interface PerkListener
    extends java.util.EventListener {
  public abstract void startedPerking (PerkEvent e);
}
```

1.7.6. Aufgaben

In dieser Übung wollen wir die relevanten Klassen in einem JAR zusammenfassen. Damit können wir die Klassendateien überall einsetzen. Zudem bauen wir eine `BeanInfo` ein.

Ihre Aufgaben im Einzelnen:

1) kreieren Sie ein neues Package bestehend aus den Perk Dateien
2) kreieren Sie eine `BeanInfo` Klasse für `CentralPerk`
3) kreieren Sie eine Manifest Datei für das Bean `jar` Archiv. Dieses kann automatisch durch das jar-Hilfsprogramm ausgeführt werden. Aber manuell kann man angeben, welche Datei eine JavaBean ist.
   Pro Datei muss eine Zeile der Form
   ```
   Name: Package/CentralPerk.class
   ```
   wobei der Slash, nicht der DOS Backslash '\' als Trennzeichen verwendet werden muss.
   Für die JavaBean Klasse muss zudem eine weitere Zeile eingefügt werden:
   ```
   Java-Bean: True
   ```
   Zwischen jedem Eintrag muss eine leere Zeile eingefügt werden.
4) Kreieren Sie aus dem übergeordneten Verzeichnis das Java Archiv.
5) Starten Sie die BeanBox.
6) Laden Sie das Java Archiv
7) Testen Sie die Bean in der BeanBox.

1.7.7. Hilfestellungen

Zu jeder der obigen Aufgaben können Sie kurze Lösungshinweise erhalten. Zudem steht Ihnen weiter unten die Musterausgabe und eine Musterlösung zur Verfügung.

1) Da wir JBuilder verwenden, sind die einzelnen Dateien der vorangehenden Übung bereits in einem Package.

2) Da unsere Bean Klasse `CentralPerk` heisst, muss die `BeanInfo` Klasse `CentralPerkBeanInfo` genannt werden. Diese muss die Klasse `SimplebeanInfo` erweitern.
   Die `getBeanDescriptor()` Methode muss folgenden Aufbau haben:

```
public BeanDescriptor getBeanDescriptor() {
  BeanDescriptor bd = new BeanDescriptor(CentralPerk.class);
  bd.setDisplayName($quot;Billboard");
  return bd;
}
```

3) Hier der Inhalt der Manifest Datei:

```
Name: packaged/CentralPerk.class
Java-Bean: True

Name: packaged/CentralPerkBeanInfo.class

Name: packaged/PerkEvent.class

Name: packaged/PerkListener.class
```

   Das Gleiche können Sie auch mit einem Hilfsprogramm wie nmake (Windows) oder make (Unix) erledigen (lassen). Damit sparen Sie viel unnötige Arbeit, speziell bei vielen Dateien.

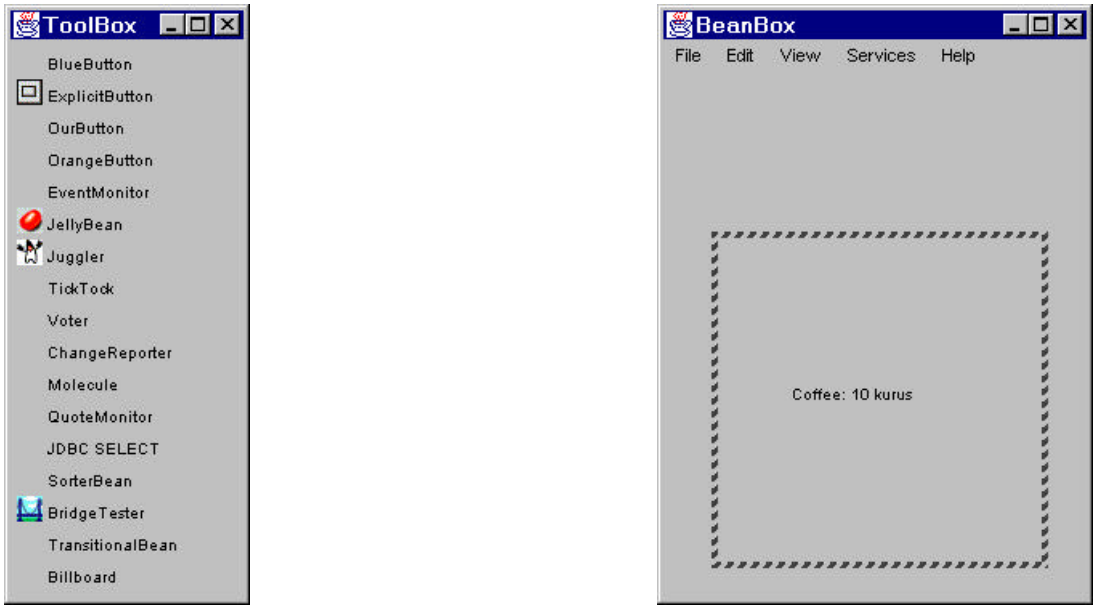
4) Falls die Manifest Datei manifest heisst, siegt der JAR Befehl folgendermassen aus:

```
jar cfm Perk.jar manifest <PACKAGE NAME>\*.class
```

5) Wie das BDK zu handhaben ist, können Sie in der ersten Übung nachlesen und sehen.

6) Falls Sie einen Namen für Ihr Archiv gewählt haben, der bereits verwendet wird, wird eine Exception geworfen.

1.7.8. Demo



Da wir unsere Bean "Billboard" genannt haben, erscheint auf der linken Seite in der ToolBox diese Bean. Auf der rechten Seite sehen Sie ein Beispiel für den Einsatz dieser Bean.

Einige Hinweise:
- beachten Sie, dass das Manifest 100% korrekt geschrieben ist. Speziell Gross- und Kleinschreibung ist kritisch.
- ich habe der Einfachheit halber das Java Archiv ins Verzeichnis der JAR's im BDK Verzeichnis verschoben, damit das Laden einfacher ist.

1.7.9. Musterlösung

Am Quellcode haben wir nicht verändert! Einzig die Manifest Datei muss erstellt werden. Hier der Inhalt der Manifest Datei im JAR Archiv:

```
Manifest-Version: 1.0
Name: packaging/CentralPerk.class
Java-Bean: True
Created-By: 1.3.0 (Sun Microsystems Inc.)

Name: packaging/PerkEvent.class

Name: packaging/CentralPerkBeanInfo.class

Name: packaging/PerkListener.class
```

Der Name der Datei wird im Archiv auf `Manifest.mf` verändert.

1.8. Arbeiten mit Ressourcen

Applets verfügen über hilfreiche Methoden, wie `getCodeBase()`, `getDocumentBase()` oder `getParameter()`. Beans sind aber selbständige Software Komponenten und benötigen einen anderen Mechanismus, das Java Resource File. Diese Datei kann entweder im Archiv mitverpackt oder in der lokalen CLASSPATH gestellt werden.

1.8.1. Lernziele

- Sie lernen die Bedeutung von Ressourcen für JavaBeans kennen
- Sie lernen Resource Files zu laden.

1.8.2. Rahmenprogramm

Für diese Übung benötigen wir lediglich eine Datei. Da wie das Rote Kreuz Symbol verwenden und Clara Barton (*Angel of the Battlefield*) das Rote Kreuz in den USA eingeführt hat, nennen wie die Datei Barton.

1.8.2.1. Barton

```
import java.io.*;
import java.net.*;
import java.awt.*;
import java.awt.event.*;
import java.util.Properties;

public class Barton extends Frame {
  Image im;
  Font f;
  String fontDefault = "Serif-bolditalic-24";
  String msg;
  String msgDefault = "Spenden Sie bitte Blut!";
  Color savedColor;
  public Barton () {
    super ("Spenden Sie Blut!");
    // Ressource Datei


    // Laden der Datei
    try {
      Properties p = new Properties();

      // Laden der Ressourcen aus der Eingabedatei / InoutStream

      // Font:'message.font'
      // sonst fontDefault


      // Message: 'message.text'
      // sonst msgDefault

      // Image: 'image.file'
      // kein Standardwert

      // Lies Image

    } catch (Exception e) {
      f = Font.decode (fontDefault);
      msg = msgDefault;
      im = null;
```

```
      System.err.println ("Exception - es werden Standardwerte
verwendet.");
    }
    setFont (f);
    // Windows Close
    enableEvents (AWTEvent.WINDOW_EVENT_MASK);
  }
  public void paint (Graphics g) {
    if (im == null) {
      savedColor = g.getColor();
      g.setColor (Color.red);
      g.fillRect (50, 25, 30, 80);
      g.fillRect (25, 50, 80, 30);
      g.setColor (savedColor);
    } else {
      g.drawImage (im, 25, 20, this);
    }
    g.drawString (msg, 50, 130);
  }
  protected void processWindowEvent (WindowEvent e) {
    if (e.getID() == WindowEvent.WINDOW_CLOSING) {
      System.exit (0);
    }
  }
  public static void main (String args[]) {
    Barton b = new Barton();
    b.setSize (300, 200);
    b.show();
  }
}
```

1.8.3. Aufgaben

Die Aufgaben sind nicht sehr schwierig, falls Sie Java kennen und wissen wie man mit Dateien umgeht, oder falls Sie die Java Internationalisierung kennen.

1) Öffnen Sie einen `InputStream` auf die Datei `Barton.resources`
2) Laden Sie die Properties
3) Bestimmen Sie die `message.font` Eigenschaft. Falls diese nicht vorhanden ist oder nicht gefunden wird, soll der Standardwert `fontDefault` verwendet werden. Diese Angabe muss in ein `Font` Objekt umgewandelt werden.
4) Bestimmen Sie die `message.text` Eigenschaft. Falls diese nicht gefunden wird, soll der Standardwert `msgDefault` verwendet werden. Dieser stellt einen einfachen Text zur Verfügung, der dargestellt werden soll.
5) Bestimmen Sie die `image.file` Eigenschaft. Sie enthält die Angabe, welche Datei engezeigt werden soll (als Bild). Öffnen Sie diese Datei und laden Sie das Bild.
6) Übersetzen und starten Sie das Programm. Da Sie noch keine Ressource Datei angegeben haben, werden lauter Standardwerte verwendet.
7) Verschieben Sie jetzt die `Barton.resources` Datei ind aktuelle Verzeichnis oder in den CLASSPATH. Auch die Datei mit dem Roten Kreuz muss im aktuellen Verzeichnis oder im CLASSPATH sein.
   Starten Sie das Programm erneut.

Hinweis:
an Stelle der Datei könnten Sie auch eine URL verwenden, eine Internet Adresse (Uniform Resource Locator), ab der die Informationen gelesen werden können.

1.8.4. Hilfestellungen

Die folgenden Hilfestellungen beziehen sich auf die einzelnen Aufgaben aus dem obigen Abschnitt.

1) Beispielcode
```
 InputStream is =
getClass().getResourceAsStream("Barton.resources");
```

2) Beispielcode
verwenden Sie einfach die `Property.load(InputStream)` Methode:
```
p.load(is);
```

Den Wert einer Property bestimmenSie am Besten mit der Methode
`getProperty()`

3) Für das Bestimmen des Fonts benötigen wir zwei Schritte:
a) bestimmen der Property, wie oben
b) `f = Font.decode(p.getProperty("message.font", fontDefault);`

4) Die Bestimmung des Textes geschieht mit der oben bereits eingesetzten `getProperty() Methode`:

```
msg = p.getProperty("message.text", msgDefault);
```

5) Das Bestimmen und Laden es Bildes geschieht ebenfalls mit den selben Methoden:

```
String imageFile = p.getProperty("image.file");
URL url = getClass().getResource(imageFile);
im = getToolkit().getImage(url);
```

6) Das Übersetzen und Starten dürfte nun kein Problem mehr sein!

```
javac Barton.java
java Barton
```

7) Die Ressource Dateien finden Sie auf dem Server / der CD.

Hinweis:
Barton ist kein Applet,sondern eine selbständige Applikation.

1.8.5. Demo

Auf der linken Seite sehen Sie die Ausgabe mit den Standardwerten.
Rechts wird die Ressourcen Datei gelesen und der entsprechende Text (und das Bild) ausgegeben.

1.8.6. Musterlösung

1.8.6.1. Barton

```
package ressourcen;

/**
 * Title:        Ressourcen
 * Description:  EInsatz einer Ressourcen Datei mit allen wichtigen
variablen Angaben.
 * Copyright:    Copyright (c) J.M.Joller
 * Company:      Joller-Voss
 * @author J.M.Joller
 * @version 1.0
 */

import java.io.*;
import java.net.*;
import java.awt.*;
import java.awt.event.*;
import java.util.Properties;

public class Barton extends Frame {
  Image im;
  Font f;
  String fontDefault = "Serif-bolditalic-24";
  String msg;
  String msgDefault = "Spenden Sie bitte Blut!";
  Color savedColor;
  public Barton () {
    super ("Spenden Sie Blut!");
    // Ressource Datei
    InputStream is = getClass().getResourceAsStream(
                      "Barton.resources");
    // Laden und Anzeigen der Ressourcen
    try {
      Properties p = new Properties();
      p.load (is);
      f = Font.decode(
         p.getProperty("message.font",
                      fontDefault));
```

```
      msg = p.getProperty ("message.text", msgDefault);
      // Image
      String imageFile = p.getProperty ("image.file");
      URL url = getClass().getResource (imageFile);
      im = getToolkit().getImage (url);
    } catch (Exception e) {
      f = Font.decode (fontDefault);
      msg = msgDefault;
      im = null;
      System.err.println ("Exception - es werden Standardwerte
verwendet.");
    }
    setFont (f);
    // Window Closing
    enableEvents (AWTEvent.WINDOW_EVENT_MASK);
  }
  public void paint (Graphics g) {
    if (im == null) {
      savedColor = g.getColor();
      g.setColor (Color.red);
      g.fillRect (50, 25, 30, 80);
      g.fillRect (25, 50, 80, 30);
      g.setColor (savedColor);
    } else {
      g.drawImage (im, 25, 20, this);
    }
    g.drawString (msg, 50, 130);
  }
  protected void processWindowEvent (WindowEvent e) {
    if (e.getID() == WindowEvent.WINDOW_CLOSING) {
      System.exit (0);
    }
  }
  public static void main (String args[]) {
    Barton b = new Barton();
    b.setSize (300, 200);
    b.show();
  }
}
```

1.8.6.2. Ressourcen Datei

```
message.font=SansSerif-bold-24
message.text=Freiwillige Spender gesucht!
image.file=redcross.gif
```

1.9. Customizer

1.9.1. Voraussetzungen

Sie haben die vorangehenden Übungen durchgearbeitet.

1.9.2. Rahmenprogramme

Für diese Übung verwenden wir verschiedene Programme aus früheren Übungen: Perk.

1) CentralPerk
2) PerkEvent
3) PerkListener
4) CentralPerkBeanInfo

Zusätzlich benötigen wir zwei weitere Dateien

1.9.2.1. Top40

```
import java.awt.*;
import java.awt.event.*;

import java.io.*;

public class Top40 extends Frame implements ActionListener {
  MenuItem open, load, save, custom, quit;
  Component theComp = null;

  public Top40 () {
    super ("Countdown");
    Menu file = new Menu ("DAtei");
    open = file.add (new MenuItem ("Öffnen Serialisierte Bean"));
    open.addActionListener (this);
    load = file.add (new MenuItem ("Laden CentralPerk Klasse"));
    load.addActionListener (this);
    save = file.add (new MenuItem ("Speichern Bean"));
    save.addActionListener (this);
    save.setEnabled (false);
    file.addSeparator();
    custom = file.add (new MenuItem ("Customize Bean"));
    custom.addActionListener (this);
    custom.setEnabled (false);
    file.addSeparator();
    quit = file.add (new MenuItem ("Ende"));
    quit.addActionListener (this);
    MenuBar mb = new MenuBar();
    mb.add (file);
    setMenuBar (mb);
    enableEvents (AWTEvent.WINDOW_EVENT_MASK);
  }
  protected void processWindowEvent(WindowEvent e) {
    if (e.getID() == WindowEvent.WINDOW_CLOSING) {
      // Notify others we are closing
      super.processWindowEvent(e);
      System.exit(0);
    } else {
      super.processWindowEvent(e);
    }
  }
  public static void main (String args[]) {
```

```
    Frame f = new Top40();
    f.setSize(300, 300);
    f.show();
  }
  public void actionPerformed (ActionEvent e) {
    Object o = e.getSource();
    if (o == open) {
      loadBean();
    } else if (o == load) {
      newBean();
    } else if (o == save) {
      saveBean();
    } else if (o == custom) {
      customize();
    } else if (o == quit) {
      System.exit (0);
    }
  }
  private void loadBean () {
    setCursor (Cursor.getPredefinedCursor(Cursor.WAIT_CURSOR));
    try {
      // Datei CentralPerk.ser

      // Stream (FileInputStream)

      // Stream (ObjectInputStream)

      // lesen

      // Alles schliessen

      // Bean in den zentralen Quadranten

      // checkCustomize() um Menu aufzusetzen

      // Referenz speichern

      save.setEnabled (true);
    } catch (Exception e) {
      System.out.println ("CentralPerk.ser kann nicht geladen werden");
    } finally {
      setCursor (Cursor.getPredefinedCursor(Cursor.DEFAULT_CURSOR));
    }
  }
  private void newBean () {
    try {
      setCursor (Cursor.getPredefinedCursor(Cursor.WAIT_CURSOR));
      // Instantieren der Bean

      // Bean dem Center Quadrant hinzufügen

      // Call checkCustomize to setup menu

      // Referenz speichern

      save.setEnabled (true);
    } catch (Exception e) {
      System.out.println ("Fehler beim Kreieren");
    } finally {
      setCursor (Cursor.getPredefinedCursor(Cursor.DEFAULT_CURSOR));
    }
  }
```

```
  private void saveBean () {
    setCursor (Cursor.getPredefinedCursor(Cursor.WAIT_CURSOR));
    try {
      // Dateiobjekt CentralPerk.ser

      // Stream für FileOutputStream

      // Stream für ObjectOutputStream

      // schreiben

      // alles schliessen

    } catch (Exception e) {
      System.out.println ("Fehler beim Schreiben");
    } finally {
      setCursor (Cursor.getPredefinedCursor(Cursor.DEFAULT_CURSOR));
    }
  }
  private void customize () {
    try {
      // BeanInfo

      // BeanDescriptor für die BeanInfo

      // Bestimme Customizer Klasse aus dem BeanDescriptor

      // Kreiere Instanz

      // Customizer definieren

      final Dialog d = new Dialog (this, "Customizer", true);
      d.add ((Component)customizer, BorderLayout.CENTER);
      Button b = new Button ("Ende");
      d.add (b, BorderLayout.SOUTH);
      b.addActionListener (new ActionListener () {
          public void actionPerformed (ActionEvent e) {
            d.dispose();
          }
        }
      );
      d.pack();
      d.show();
    } catch (Exception e) {
      System.out.println ("Oops");
    }
  }
  private void checkCustomize(Class c) {
    try {
      BeanInfo bi = Introspector.getBeanInfo (c);
      BeanDescriptor bd = bi.getBeanDescriptor();
      custom.setEnabled (bd.getCustomizerClass() != null);
    } catch (IntrospectionException e) {
      System.out.println ("Oops");
      custom.setEnabled (false);
    }
  }
}
```

1.9.2.2. BillboardCustomizer

```
package Marcel;

import java.awt.*;
import java.awt.event.*;
import java.beans.*;

public class BillboardCustomizer extends Panel {
  private CentralPerk target;
  private TextField fontField, msgField;
  private PropertyChangeSupport support =
    new PropertyChangeSupport(this);
  public void setObject(Object obj) {
    target = (CentralPerk) obj;
    // Font und Message EIngabefelder
    // Action Listeners
  }
  public Dimension preferredSize() {
    return new Dimension(225, 60);
  }
  public void actionPerformed(ActionEvent e) {
    Object source = e.getSource();
    if (source==fontField) {
      // Font anpassen

      support.firePropertyChange("", null, null);
    }else if (source==msgField) {
      // Message anpassen

      support.firePropertyChange("", null, null);
    }
  }
  public void addPropertyChangeListener(
      PropertyChangeListener l) {
    support.addPropertyChangeListener(l);
  }
  public void removePropertyChangeListener(
      PropertyChangeListener l) {
    support.removePropertyChangeListener(l);
  }
  private String encode (Font f) {
    String style;
    if (f.isBold())
      style = f.isItalic() ? "bolditalic" : "bold";
    else
      style = f.isItalic() ? "italic" : "plain";
    return f.getName() + "-" + style + "-" + f.getSize();
  }
}
```

1.9.3. Aufgaben

In dieser Übung ergänzen Sie die CentralPerk Billboard Bean durch einen Customizer. Zudem benötigen Sie eine Testapplikation. Diese kann die Bean laden und speichern.

1) verwenden Sie die Rahmenprogramem aus dieser Übung, neben den fix fertigen Dateien aus den vorhergehenden Übungen. Diese Dateien werden zu einem neuen Package zusammengefasst. Deswegen wurden die Dateien ins JBuilder Projekt kopiert.

2) Unser Tester entspricht einer Anwendung, die sich analog zur BeanBox verhält. Es liegt an uns, einen Customizer zu definieren, um die Bean Properties / Eigenschaften zu aktualisieren.

   Wir haben zwei Möglichkeiten:
   - entweder einen Customizer für die Bohne definieren
   - oder ein Property Sheet für Font, Message Eigenschaften (Text, Font, ...)

   Das Property Sheet lassen wir weg. Es bleibt also der Customizer.

3) Verändern Sie die BeanInfo so, dass diese den Customizer kennt.

4) Übersetzen Sie alles, ohne ein Java Archiv zu generieren, ausser Sie möchten dies.

5) Im Top40 Framework werden vier Methoden zur Verfügung gestellt:
   - `newBean`: diese kreiert eine neue Bohne, eine `CentralPerk` Bean.
   Die Bean soll dem zentralen Quadranten hinzugefügt werden.
   Die `checkCustomizer()` Methode generiert ein Menü, falls ein Customizer existiert.
   Speichern Sie eine Referenz auf die Bohne in der Variable `theComp`.

6) Die nächste Methode
   - `saveBean` : speichert die Bohne, serialisiert als `centralPerk.ser`.

7) Die nächste Methode
   - `loadBean`: liest eine (serialisierte) Bohne und speichert eine Referenz in `theComp`.

8) Schliesslich
   - `CustomizeBean` Menü: zeigt den Customizer an.
   Inspector liefert `BeanInfo`, dann `BeanDescriptor`, dann `CustomizerClass`.

9) Übersetzen Sie `Top40`

10) Starten Sie das Programm und laden Sie die CentralPerk Bean. Diese können Sie dann speichern / serialisieren, anpassen, laden, ...

1.9.4. Hilfestellungen

Auch in dieser Übung steht pro obige Aufgabe eine einfache Hilfestellung zur Verfügung. Zudem sehen Sie die erwartete Ausgabe (nächster Abschnitt) und eine mögliche Musterlösung.

1) Rahmenprogramme
   In der Musterlösung wurde das gesamte Perk Projekt ins neue Projekt kopiert und angepasst (Package Name). Zudem wurde ein neues JAR Archive generiert, die Manifest Datei angepasst und ins BDK geladen und schliesslich serialisiert.

2) Customizer
   Das Schreiben des Customizers selber ist nicht sehr aufwendig, falls Sie das Rahmenprogramm verwenden.
   - Als Customizer wird ein separates Fenster geöffnet, in dem Sie die Schrift und die Schriftgrösse der Anzeige auf der Tafel (Billboard) steuern können.

   - Der Customizer besitzt einen "Done/Beenden" Knopf, für den Sie einen Listener definieren müssen. Die Musterlösung ist dabei eher bieder!

   - Der Grund, weshalb die Musterlösung bieder ist, liegt vorallem an der `actionPerformed()` Methode. Besser wäre es auch, wenn eine Liste der verfügbaren Fonts und deren Grössen engezeigt werden könnte.

3) Die BeanInfo kann aus dem Customizer durch Änderung der `getBeanDescriptor()` Methode erreicht werden:

```
public BeanDescriptor getBeanDescriptor() {
  BeanDescriptor bd = new BeanDescriptor(CentralPerk.class,
                                  BillboardCustomizer.class);
...}
```

4) Übersetzen und Packaging (falls gewünscht) geschieht genau wie in der vorigen Übung.

5) Jetzt widmen wir uns dem Top40 Framework:
   die verschiedenen Menüoptionen müssen implementiert werden.
   Bei den Dateien ist der Pfad wichtig, weil wir der Einfachheit halber mit absoluten Pfadangaben arbeiten.

   Hier ein möglicher Programmteil, um die serialisierte Bohne in einen Container zu stellen:

```
Component c = (Component)Beans.instantiate(null,
                        "customizer.CentralPerk");
add(c, BorderLayout.CENTER);
checkCustomizer(c.getClass() );
dieComponente = c;
```

6) Das Speichern der Bohne wird als Objektserialisierung implementiert.
Hier ein Programmfragment:

```
File f = new File("CentralPerk.ser");
FileOutputStream fos = new FileOutputStream(f);
ObjectOutputStream oos = new ObjectOutputStream(fos);
oos.writeObject(dieComponente);
oos.close();
fos.close();
```

7) Das Lesen der Bohne geschieht in die umgekehrte Richtung.
Hier ein Programmfragment:

```
File f = new File ("CentralPerk.ser");
FileInputStream fis = new FileInputStream(f);
ObjectInputStream ois = new ObjectInputStream (fis);
Component c = (Component)ois.readObject();
add (c, BorderLayout.CENTER);
ois.close();
fis.close();
checkCustomize(c.getClass());
dieComponente = c;
```

8) Das Anpassen der BeanInfo geschieht mit folgendem Programmfragment:

```
BeanInfo bi = Introspector.getBeanInfo (theComp.getClass());
BeanDescriptor bd = bi.getBeanDescriptor();
Class c = bd.getCustomizerClass();
Customizer customizer = (Customizer) c.newInstance();
customizer.setObject (theComp);
```

9) Übersetzen Sie Top40

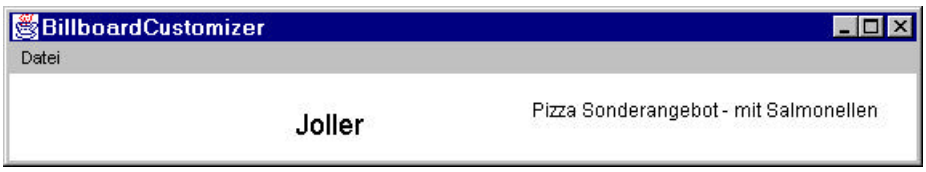
10) Testen Sie Top40
Die Änderungen im Customizer der Musterlösung werden erst beim erneuten Laden sichtbar.

1.9.5. Demo

Hier einige ScreenShots von unseren Anzeigetafel:

Die Anzeigetafel nach dem Start und nach dem Laden der serialisierten Bean.

Das Customizer Fenster



... und der geänderte Text (die Musterlösung löscht den alten Text nicht).

Nach dem Laden der zuvor serialisierten Bean sehen Sie folgendes Bild:

1.9.6. Musterlösung

Die Musterlösung auf dem Server / der CD enthält alle Dateien:

- Top40.java
- BillboardCustomizer.java
- CentralPerk.java
- CentralPerkBeanInfo.java
- PerkEvent.java
- PerkListener.java

Für diese Aufgabe sind die beiden Programme `Top40` und `BillboardCustomizer` die interessanten.

1.9.6.1. Top40

```
package customizer;

/**
 * Title:        Einfacher Customizer
 * Description:  Bau eines einfachen Customizers für ein bestehendes
Projekt
 * Copyright:    Copyright (c) J.M.Joller
 * Company:      Joller-Voss
 * @author J.M.Joller
 * @version 1.0
 */

import java.awt.*;
import java.awt.event.*;
import java.beans.*;
import java.io.*;

public class Top40 extends Frame implements ActionListener {
  MenuItem open, load, save, custom, quit;
  Component theComp = null;

  public Top40 () {
    super ("BillboardCustomizer");
    Menu file = new Menu ("Datei");
    open = file.add (new MenuItem ("Öffnen Serialisierte Bean"));
    open.addActionListener (this);
    load = file.add (new MenuItem ("Laden Central Perk Klasse"));
    load.addActionListener (this);
    save = file.add (new MenuItem ("Speichern Bean"));
    save.addActionListener (this);
    save.setEnabled (false);
    file.addSeparator();
    custom = file.add (new MenuItem ("Customize Bean"));
    custom.addActionListener (this);
    custom.setEnabled (false);
    file.addSeparator();
    quit = file.add (new MenuItem ("Ende"));
    quit.addActionListener (this);
    MenuBar mb = new MenuBar();
    mb.add (file);
    setMenuBar (mb);
    enableEvents (AWTEvent.WINDOW_EVENT_MASK);
  }
  protected void processWindowEvent(WindowEvent e) {
```

```
    if (e.getID() == WindowEvent.WINDOW_CLOSING) {
      // Notify others we are closing
      super.processWindowEvent(e);
      System.exit(0);
    } else {
      super.processWindowEvent(e);
    }
  }
  public static void main (String args[]) {
    Frame f = new Top40();
    f.setSize(300, 300);
    f.show();
  }
  public void actionPerformed (ActionEvent e) {
    Object o = e.getSource();
    if (o == open) {
      loadBean();
    } else if (o == load) {
      newBean();
    } else if (o == save) {
      saveBean();
    } else if (o == custom) {
      customize();
    } else if (o == quit) {
      System.exit (0);
    }
  }
  private void loadBean () {
    setCursor (Cursor.getPredefinedCursor(Cursor.WAIT_CURSOR));
    try {
      File f = new File ("CentralPerk.ser");
      FileInputStream fis = new FileInputStream(f);
      ObjectInputStream ois = new ObjectInputStream (fis);
      Component c = (Component)ois.readObject();
      add (c, BorderLayout.CENTER);
      ois.close();
      fis.close();
      checkCustomize(c.getClass());
      theComp = c;
      save.setEnabled (true);
    } catch (Exception e) {
      System.out.println ("Fehler beim Laden: CentralPerk.ser");
      e.printStackTrace();
    } finally {
      setCursor (Cursor.getPredefinedCursor(Cursor.DEFAULT_CURSOR));
    }
  }
  private void newBean () {
    try {
      setCursor (Cursor.getPredefinedCursor(Cursor.WAIT_CURSOR));
      Component c = (Component)Beans.instantiate (null,
"customizer.CentralPerk");
      add (c, BorderLayout.CENTER);
      checkCustomize(c.getClass());
      theComp = c;
      save.setEnabled (true);
    } catch (Exception e) {
      System.out.println ("Fehler beim Kreieren von
customizer.CentralPerk");
    } finally {
      setCursor (Cursor.getPredefinedCursor(Cursor.DEFAULT_CURSOR));
    }
```

```
  }
  private void saveBean () {
    setCursor (Cursor.getPredefinedCursor(Cursor.WAIT_CURSOR));
    try {
      File f = new File ("CentralPerk.ser");
      FileOutputStream fos = new FileOutputStream(f);
      ObjectOutputStream oos = new ObjectOutputStream (fos);
      oos.writeObject (theComp);
      oos.close();
      fos.close();
    } catch (Exception e) {
      System.out.println ("Schreibfehler: CentralPerk.ser");
    } finally {
      setCursor (Cursor.getPredefinedCursor(Cursor.DEFAULT_CURSOR));
    }
  }
  private void customize () {
    try {
      BeanInfo bi = Introspector.getBeanInfo (theComp.getClass());
      BeanDescriptor bd = bi.getBeanDescriptor();
      Class c = bd.getCustomizerClass();
      Customizer customizer = (Customizer) c.newInstance();
      customizer.setObject (theComp);
      final Dialog d = new Dialog (this, "Customizer", true);
      d.add ((Component)customizer, BorderLayout.CENTER);
      Button b = new Button ("Beenden");
      d.add (b, BorderLayout.SOUTH);
      b.addActionListener (new ActionListener () {
          public void actionPerformed (ActionEvent e) {
            d.dispose();
          }
        }
      );
      d.pack();
      d.show();
    } catch (Exception e) {
      System.out.println ("Oops");
    }
  }
  private void checkCustomize(Class c) {
    try {
      BeanInfo bi = Introspector.getBeanInfo (c);
      BeanDescriptor bd = bi.getBeanDescriptor();
      custom.setEnabled (bd.getCustomizerClass() != null);
    } catch (IntrospectionException e) {
      System.out.println ("Oops");
      custom.setEnabled (false);
    }
  }
}
```

1.9.6.2. BillboardCustomizer

```
package customizer;
import java.awt.*;
import java.awt.event.*;
import java.beans.*;
public class BillboardCustomizer extends Panel
    implements Customizer, ActionListener {
  private CentralPerk target;
  private TextField fontField, msgField;
  private PropertyChangeSupport support =
    new PropertyChangeSupport(this);
  public void setObject(Object obj) {
    target = (CentralPerk) obj;
    Label t1 = new Label("Font:");
    add(t1);
    fontField = new TextField(encode(target.getFont()), 20);
    add(fontField);
    fontField.addActionListener(this);
    Label t2 = new Label("Message:");
    add(t2);
    msgField = new TextField(target.getMessage(), 20);
    add(msgField);
    msgField.addActionListener(this);
  }
  public Dimension preferredSize() {
    return new Dimension(225, 60);
  }
  public void actionPerformed(ActionEvent e) {
    Object source = e.getSource();
    if (source==fontField) {
      String txt = fontField.getText();
      try {
        target.setFont (Font.decode (txt));
      } catch (NumberFormatException ex) {
        fontField.setText(encode(target.getFont()));
      }
      support.firePropertyChange("", null, null);
    }else if (source==msgField) {
      target.setMessage (msgField.getText());
      support.firePropertyChange("", null, null);
    }
  }
  public void addPropertyChangeListener(
      PropertyChangeListener l) {
    support.addPropertyChangeListener(l);
  }
  public void removePropertyChangeListener(
      PropertyChangeListener l) {
    support.removePropertyChangeListener(l);
  }
  private String encode (Font f) {
    String style;
    if (f.isBold())
      style = f.isItalic() ? "bolditalic" : "bold";
    else
      style = f.isItalic() ? "italic" : "plain";
    return f.getName() + "-" + style + "-" + f.getSize();
  }
}
```

1.10. Einsatz des Reflection APIs

Diese Übung ist optional.
Falls Sie je komplexere Anwendungen schreiben wollen, sollten Sie sich mit den Details des Reflection APIs und der Introspection für JavaBeans vertraut machen.

1.10.1. Lernziele

- Sie lernen den Einsatz des Reflection APIs in JavaBeans kennen
- Sie kennen den Einsatz der Introspection für die Bestimmung der Beans Eigenschaften ohne BeanInfo.

1.10.2. Rahmenprogramm

Für diese Übung benötigen Sie lediglich ein Programm:

```
import java.io.*;


public class ReflectionBeispiel {

  private static String className = "RefelctionBeispiel";
  private static String getTypeName (Class c) {
    if (c.isArray()) {
      try {
        Class cl = c;
        int dimensions = 0;
        while (cl.isArray()) {
          dimensions++;
          cl = cl.getComponentType();
        }
        StringBuffer sb = new StringBuffer();
        sb.append(cl.getName());
        for (int i = 0; i < dimensions; i++) {
          sb.append("[]");
        }
        return sb.toString();
      } catch (Throwable e) {
         // keine Ausgabe
      }
    }
    return c.getName();
  }

  private static void listParameters (PrintWriter pw, Class parameters[]) {
    for (int j=0; j< parameters.length; j++) {
      pw.print (getTypeName(parameters[j]));
      if (j < (parameters.length - 1))
        pw.print(", ");
    }
  }

  private static void listClassVariables (PrintWriter pw, Class c) {
    String name = c.getName();
    // Ausgabe der Variablen
    Field fields[] = c.getDeclaredFields();
    for (int i=0; i<fields.length;i++) {
      if (i == 0)
        pw.println ("  // Variablen");
```

```
      // nur jene der Klasse
      pw.println ("  " + Modifier.toString(fields[i].getModifiers()) +
                  " " + getTypeName (fields[i].getType()) +
                  " " + fields[i].getName() + ";");
    }
    if (fields.length > 0)
      pw.println();
  }

  private static void listClassConstructors (PrintWriter pw, Class c) {
    String name = c.getName();
    // Ausgabe der Konstruktoren
    Constructor constructors[] = c.getDeclaredConstructors();
    for (int i=0; i<constructors.length; i++) {
      if (i==0)
        pw.println ("  // Konstruktoren");
      pw.print ("  " + Modifier.toString(constructors[i].getModifiers()) +
                   " " + constructors[i].getName() + "(");
      listParameters (pw, constructors[i].getParameterTypes());
      pw.println (");");
    }
    if (constructors.length > 0)
      pw.println();
  }

  private static void listClassMethods (PrintWriter pw, Class c) {

    //************************
    // Ihre Aufgabe!!!
    //************************

  }

  public static void reflectClass (String name, Writer w) {
    PrintWriter pw = new PrintWriter (w);

    // Instanzieren der Klasse
    Class c;
    try {
      c = Class.forName(name);
    } catch (Exception e) {
      pw.println ("Klasse " + name + " wurde nicht gefunden.");
      return;
    }

    // Ausgabe der Deklarationen
    pw.println (Modifier.toString(c.getModifiers()) + " " + c.getName());
    // Ausgabe der Superklassen
    if (c.getSuperclass() != null)
      pw.print ("  extends " + c.getSuperclass().getName());
    // Ausgabe der Interfaces
    Class interfaces[] = c.getInterfaces();
    for (int i=0; i<interfaces.length; i++) {
      if (i == 0) {
        pw.print (" implements ");
      } else {
        pw.print (", ");
      }
      pw.print (interfaces[i].getName());
    }
    pw.println (" {");
```

```
    listClassVariables (pw, c);
    listClassConstructors (pw, c);
    listClassMethods (pw, c);

    pw.println ("}");
    pw.flush();
  }

  public static void reflectClass (String name, OutputStream os) {
    Writer w = new OutputStreamWriter (os);
    reflectClass (name, w);
  }

  private static void inspectSelf () {
    reflectClass (className, System.out);
  }

  public static void main (String args[]) {
    if (args.length == 0) {
      System.err.println ("Usage: java " + className + " classname [
classname]*\n");
      inspectSelf();
    } else {
      for (int i=0;i<args.length;i++) {
        reflectClass (args[i], System.out);
      }
    }
  }
}
```

1.10.3. Aufgaben

Zu den folgenden Aufgaben finden Sie gleich anschliessend Hilfestellungen und Demo Ausgaben, sowie eine vollständige Musterlösung.

1) Importieren Sie die Reflection Library
2) Im Rahmenprogramm werden die Klassenvariablen und Konstruktoren jeder benötigten Klasse bestimmt.
   Modifizieren Sie das Programm so, dass auch die Methoden aufgelistet werden. Im Framework finden Sie bereits einiges. Eigentlich müssen Sie lediglich die `listClassMethods()` Methode hinzufügen.
3) Versuchen Sie zudem, neben dem Namen, auch noch die Modifiers, Rückgabetypen und die Parametertypen zu bestimmen und anzuzeigen.

Zusatzaufgaben:

1) sortieren Sie die einzelnen Listen
2) gruppieren Sie die einzelnen Methoden und Variablen nach Zugriffstyp (`private, public, protected`)
3) entfernen Sie den jeweiligen Header, falls er `java.lang` ist.
4) definieren Sie Exception Handlers so, dass das Stack Trace jeweilis nicht angezeigt wird.
5) zeigen Sie nur Methoden an, welche über eine Parameterliste verfügen.

1.10.4. Hilfestellungen

Pro Aufgabe finden Sie eine einfache Hilfestellung. Falls diese nicht reicht, können Sie entweder das Beispielverhalten (Demo) oder die Musterlösung anschauen.

1) Import des Reflection APIs:

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

2) Der entsprechende Methodenaufruf ist:

   ```
   Class.getDeclaredMethods()
   ```

   Diese Methode zeigt auch die geerbten Methoden an.

3) Die entsprechende Methode zum Auflisten aller Methoden ist:

   ```
   Method.getModifiers()
   ```

   Die anderen Angaben erhalten Sie mit folgenden Methoden:

   `Method.getReturnType()` zeigt die Rückgabewerte der Methoden;
   `Method.getName()` liefert den Namen der Methode;
   `Method.getParameterTypes()` liefert die Parameterklassen;
   `Method.getExceptionTypes()` schliesslich die Exceptions einer Methode.

4) Die toString() Methode ist nicht sehr nützlich, weil sie immer noch den Präfix liefert.

5) Die `private getTypeName()` Methode zeigt Ihnen die unterschiedlichen Parametertypen an.

1.10.5. Demo

Die Musterlösung ist eine Applikation, kann also nicht als Applet verwendet werden:

```
public reflectionbeispiel.ReflectionAnwendung

  extends java.lang.Object {

  // Variablen

  private static java.lang.String className;


  // Konstruktoren

  public reflectionbeispiel.ReflectionAnwendung();


  // Methoden

  public static void main(java.lang.String[]);

  private static java.lang.String getTypeName(java.lang.Class);

  private static void listParameters(java.io.PrintWriter,
java.lang.Class[]);

  private static void listClassVariables(java.io.PrintWriter,
java.lang.Class);

  private static void listClassConstructors(java.io.PrintWriter,
java.lang.Class);

  private static void listClassMethods(java.io.PrintWriter,
java.lang.Class);

  public static void reflectClass(java.lang.String, java.io.OutputStream);

  public static void reflectClass(java.lang.String, java.io.Writer);

  private static void inspectSelf();

}

Usage: java reflectionbeispiel.ReflectionAnwendung classname [ classname]*
```

1.10.6. Musterlösung

Die folgende Musterlösung zeigt eine mögliche Lösung. Diese besitzt noch einige Macken, wie Sie leicht selber feststellen können.

```
package reflectionbeispiel;

/**
 * Title:        Beispiel für den Einsatz des Reflection APIs
 * Description:  Reflection API für die Analyse der Beans
 * Copyright:    Copyright (c) J.M.Joller
 * Company:      Joller-Voss
 * @author J.M.Joller
 * @version 1.0
 */
import java.io.*;
import java.lang.reflect.*;

public class ReflectionAnwendung {

  private static String className =
"reflectionbeispiel.ReflectionAnwendung";
  private static String getTypeName (Class c) {
    if (c.isArray()) {
      try {
        Class cl = c;
        int dimensions = 0;
        while (cl.isArray()) {
          dimensions++;
          cl = cl.getComponentType();
        }
        StringBuffer sb = new StringBuffer();
        sb.append(cl.getName());
        for (int i = 0; i < dimensions; i++) {
          sb.append("[]");
        }
        return sb.toString();
      } catch (Throwable e) {
         // keine Fehlermeldung
      }
    }
    return c.getName();
  }

  private static void listParameters (PrintWriter pw, Class parameters[]) {
    for (int j=0; j< parameters.length; j++) {
      pw.print (getTypeName(parameters[j]));
      if (j < (parameters.length - 1))
        pw.print(", ");
    }
  }

  private static void listClassVariables (PrintWriter pw, Class c) {
    String name = c.getName();
    // Ausgabe der Variablen
    Field fields[] = c.getDeclaredFields();
    for (int i=0; i<fields.length;i++) {
      if (i == 0)
        pw.println ("  // Variablen");
      // nur jene der Klasse
```

```
      pw.println ("  " + Modifier.toString(fields[i].getModifiers()) +
                  " " + getTypeName (fields[i].getType()) +
                  " " + fields[i].getName() + ";");
    }
    if (fields.length > 0)
      pw.println();
  }

  private static void listClassConstructors (PrintWriter pw, Class c) {
    String name = c.getName();
    // Ausgabe der Konstruktoren
    Constructor constructors[] = c.getDeclaredConstructors();
    for (int i=0; i<constructors.length; i++) {
      if (i==0)
        pw.println ("  // Konstruktoren");
      pw.print ("  " + Modifier.toString(constructors[i].getModifiers()) +
                  " " + constructors[i].getName() + "(");
      listParameters (pw, constructors[i].getParameterTypes());
      pw.println (");");
    }
    if (constructors.length > 0)
      pw.println();
  }

  private static void listClassMethods (PrintWriter pw, Class c) {
    String name = c.getName();
    // Ausgabe der Methoden
    Method methods[] = c.getDeclaredMethods();
    for (int i=0; i<methods.length;i++) {
      if (i==0)
        pw.println ("  // Methoden");
      pw.print ("  " + Modifier.toString(methods[i].getModifiers()) +
                " " + getTypeName (methods[i].getReturnType()) +
                " " + methods[i].getName() + "(");
      listParameters (pw, methods[i].getParameterTypes());
      pw.println (");");
    }
  }

  public static void reflectClass (String name, Writer w) {
    PrintWriter pw = new PrintWriter (w);

    // Klasseninstanz
    Class c;
    try {
      c = Class.forName(name);
    } catch (Exception e) {
      System.err.println ("Klasse " + name + " wurde nicht gefunden.");
      return;
    }

    // Ausgabe der Deklarationen
    pw.println (Modifier.toString(c.getModifiers()) + " " + c.getName());
    // Ausgabe der Superklassen
    if (c.getSuperclass() != null)
      pw.print ("  extends " + c.getSuperclass().getName());
    // Ausgabe der Interfaces
    Class interfaces[] = c.getInterfaces();
    for (int i=0; i<interfaces.length; i++) {
      if (i == 0) {
        pw.print (" implements ");
      } else {
```

```
        pw.print (", ");
      }
      pw.print (interfaces[i].getName());
    }
    pw.println (" {");

    listClassVariables (pw, c);
    listClassConstructors (pw, c);
    listClassMethods (pw, c);

    pw.println ("}");
    pw.flush();
  }

  public static void reflectClass (String name, OutputStream os) {
    Writer w = new OutputStreamWriter (os);
    reflectClass (name, w);
  }

  private static void inspectSelf () {
    //System.out.println("Klassennamen: "+className);
    reflectClass (className, System.out);
  }

  public static void main (String args[]) {
    if (args.length == 0) {
      System.err.println ("Usage: java " + className + " classname [
classname]*\n");
      inspectSelf();
    } else {
      for (int i=0;i<args.length;i++) {
        reflectClass (args[i], System.out);
      }
    }
  }
}
```

Wissen Sie noch, wie Sie eine mit Reflection bestimmte Methode ausführen können? Wenn nicht: lesen Sie nochmals die Kurseinheit über Reflection durch. Diese enthält auch Beispiele für Methodenaufrufe.

1.11. Signaturen - Ein Bean mit einer Signatur versehen

Auch diese Übung ist optional.

ACHTUNG:
Sun hat die Tools geändert. Anstelle von JavaKey wird nun KeyTool und JarSigner verwendet. Eine neue version dieser Übung steht nächstens bereit.

Falls Sie Beans verkaufen wollen, sollten Sie sich darüber bewusst sein, dass die Security wegen der Download Möglichkeiten von JavaBeans über das Internet eine grosse Rolle spielt. Für jeden einzelnen Benutzer wird eine eindeutige Kennung, ein Schlüssel benötigt, beispielsweise ein public X.509 Schlüssel. Dieser wird in irgend einer Datenbank verwaltet. (Sie könnten auch komplexere Sicherheitsmechanismen mit dem Security API aufbauen).

In dieser Übung werden Sie durch das Signieren eines Applets geführt, Schritt für Schritt! Als Basis verwenden Sie dazu das Ergebnis aus der vorigen Übung.

1.11.1. Lernziele

- Sie lernen JAR Dateien / Beans zu signieren
- Sie lernen privilegierte Beans zu kreieren.

1.11.2. Voraussetzungen

Sie haben die vorangehende Übung durchgearbeitet.

1.11.3. Ihre Aufgaben

1) Prüfen Sie zuerst, ob der in der folgenden Übung eingesetzte Schlüssel bei Ihnen schon verwendet wird. In diesem Fall müssten Sie den Schlüssel der Übung umbenennen.

   ```
   javakey -r "John Hancock"
   ```

2) Testen Sie nun das unsignierte Applet mit dem `StandardTester.html` und `Appletviewer`

3) Starten Sie den Introspektor mit dem Knopf oder der Eingabetaste. Sie sollten bei Eingabe einer beliebigen Klasse deren Methoden ... sehen (wie bei der vorherigen Übung: mittels des Reflection APIs).

4) Falls Sie nun den SigniertenTester.html mit dem Appletviewer verwenden und beispielsweise java.lang.String eingeben, werden Sie in den Standard Ausgabekanal eine lange Sicherheitsfehlermeldung erhalten.

   Im Prinzip besagt diese, dass das Applet nicht vertrauenswürdig ist. Damit es vertrauenswürdig wird, muss es signiert werden. Beim Gebrauch muss dann der passende Schlüssel eingegeben werden.

   Wie das Archiv signiert werden kann, steht in den nächsten Aufgaben

5) Ein Applet kann vertrauenswürdig gemacht werden, indem ein Zertifikat generiert wird, ein JAR Archive erstellt wird, dieses signiert wird und der HTML Seite mitgeteilt wird, dass das Zertifikat benutzt wird.

   Die Generierung des Zertifikates geschieht in folgenden vier Schritten:

   a) identifizieren Sie einen Unterschriftsberechtigten und trauen Sie ihm:

   ```
   javakey -cs "John Hancock" true
   ```

   `javakey` ist das Sicherungsprogramm
   `-c` zeigt an, dass eine Identität zu kreieren ist (create)
   `-s` bedeutet, dass wir einen Code 'Signer' kreieren.
   "John Hancock" ist der Unterschriftsberechtigte
   `true` zeigt an, dass wir John Hancock trauen.

   b) nun generieren wir ein `public / private` Schlüsselpaar.

   ```
   javakey -gk "John Hancock" DAS 512 Hancock_pub Hancock_priv
   ```

   - `gk` : generiere Schlüsselpaar (generate key)
   - `"John Hancock"` : die Identität
   - `DAS` : der Algorithmus
   - `512` : die Schlüssellänge
   - `Hancock_pub` und `Hancock_priv` sind die Dateien.

   c) definieren Sie ein Profil
   Das Profil legt fest, was wie lange gültig ist.
   Diese Datei können Sie einfach übernehmen: `Hancock.cert`.

   d) generieren Sie ein Zertifikat

   ```
   javakey -gc Hancock.cert
   ```

   - gc : generiere Zertifikat (generate certificate)
   Hancock.cert : Profil für das Zertifikat

   Ergebnis : `Hancock.x509` , also die Datei, die im `out.file` Directive angegeben wurde.

6) Nun haben wir alle Vorarbeiten abgeschlossen.
   In unserem Fall müssen wir nun die Klassendatei ReflectionAnwendung.class in ein Archiv verpacken:

   ```
   jar cf ReflectionAnwendung.jar ReflectionAnwendung.class
   ```

7) Signieren des Archives:
   Unser Archiv wird mit der Datei `Hancock.sign` signiert.
   Dies geschieht mit fogendem Befehl:

   ```
   javakey -gs Hancock.sign ReflectionAnwendung.jar
   ```

   Damit wird die Datei `ReflectionAnwendung.jar.sig` kreiert.
   Diese müssen Sie umbenennen: `signedReflectionAnwendung.jar`.

   ```
   ren ReflectionAnwendung.jar.sig signedReflectionAnwendung.jar
   ```

8) HTML Seiten
   Nach all diesen Vorarbeiten und Signaturen, benötigen wir nur noch HTML Seiten, um unsere signierte Anwendung zu starten:
   `signedReflectionAnwendung.jar`.

   Diese Seiten müssen folgende Parameter im Applet Tag aufweisen:

   ```
   <APPLET archive=signedFreud.jar code=Declaration width=400
   height=400></APPLET>
   ```

9) Anleitung für einen Benutzer dieses signierten Applets:

   Als erstes benötigt der Benutzer die `Hancock.x509` Datei.
   Dann muss der Unterschreiber als vertrauenswürdig bekannt gegeben werden.
   Schliesslich muss die Zertifikationsdatei ins System importiert werden.

   Das Entfernen einer Identität aus der lokalen Vertrauensbasis geschieht mit folgendem Befehl:

   ```
   javakey -r "John Hancock"
   ```

   Der Benutzer benötigt die Datei `Hancock.c509`
   Registriert "John Hancock" als vertrauenswürdig:
   `javakey -c "John Hancock" true`

   Hier benötigen wir kein -s Flag, weil wir kein Zertifikat generieren müssen.
   Nun müssen wir die Zertifikationsdatei importieren:

   ```
   javakey -ic "John Hancock" Hancock.x509
   ```

   Und schliesslich können wir das Applet starten

1.11.4. Hilfestellungen

Wie immer steht Ihnen zu den obigen Aufgaben eine kurze Hilfestellung zur Verfügung. Falls Sie damit noch nicht zurecht kommen, finden Sie weiter unten Beispielausgaben.

1) Überprüfen, ob der Schlüssel bereits vergeben ist:
   Falls alles okay ist, sollte eine Meldung
   `"No one named John Hancock in the system"`
   Falls der Name bereits vergeben ist:
   `"Removed John Hancock"`

2) Testen:
   Sie können gleich mit der Musterlösung das Verhalten signierter Lösungen untersuchen. Dazu stehen Ihnen Batch Dateien zur Verfügung.
   Falls Sie das Archiv auspacken wollen:
   `jar xf AllesSigniert.jar`

   und der `Appletviewer:`
   `appletviewer StandardTester.html`

3) Zu dieser Aufgabe kann man keine Hilfestellung geben: klicken Sie auf den Knopf!

4) Auch zu dieser Aufgabe enthält die Aufgabenstellung bereits genügend Hinweise.

5) Diese Aufgabe können Sie lösen, indem Sie genau so vorgehen, wie in der Aufgabenstellung beschrieben. Zur Sicherheit stehen Ihnen auch hier Batch Dateien zur Verfügung.

6) Kreieren eines Archives:
   Hier der assende Befehl

   ```
   %JAVA_HOME%\bin\jar -cf AllesSigniert.jar  ReflectionAnwendung.class
   ```

   Den Inhalt können Sie mit
   `%JAVA_HOME%\bin\jar -tf AllesSigniert.jar`

   überprüfen.

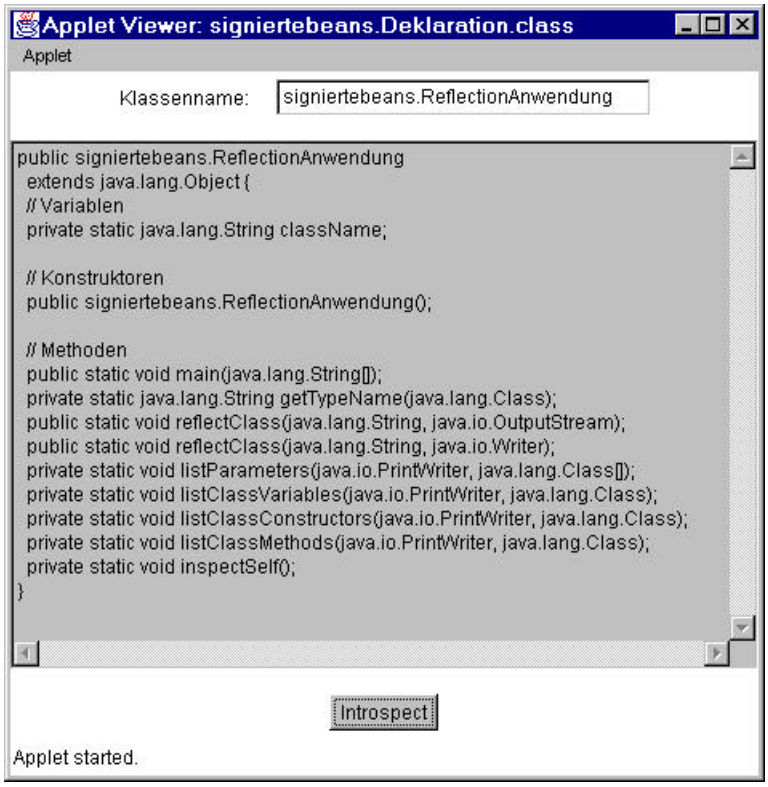
7) Signieren des Archives
   Dieser Schritt, im Detail oben beschrieben, generiert die Dateien
   `META-INF/MANIFEST.INF`
   `META-INF/HANCOCK.SF`
   `META-INF/HANCOCK.DSA`

8) Applet
   Die Lösung finden Sie bereits in der Aufgabenstellung

9) Verteilung:
   Eventuell verwenden Sie unterschiedliche Namen; aber im Grund genommen sollten Sie keinerlei Probleme haben.

1.11.5. Demo

Hier die Ausgabe des noch nicht zertifizierten Applets, mit Hilfe des `StandardTester.html`



1.11.6. Musterlösung

Da diese Aufgabe keine Java Programme enthält bzw. keine veränderte, gibt es kaum etwas zu sehen. Sie finden auf dem Server / der CD die Batch Dateien, welche Ihnen Hinweise geben sollten, wie die Aufgaben gelöst wurden.

JAVABEANS ÜBUNGEN