```
  1: import sale.*;
  2: import data.ooimpl.*;
  3: import data.*;
  4:
  5:
  6: /**
  7:  * StoringStockItem-Implementierung f&uuml;r die Videokassette im
  8:  * Bestand des Kunden.
  9:  */
 10: public class CassetteStoringStockItem extends StockItemImpl
 11: {
 12:
 13:   //// attributes ////////////////////////////////////////////////////////
 14:
 15:   private Object date;                       // Ausleihdatum
 16:
 17:   //// constructor ///////////////////////////////////////////////////////
 18:
 19:   /**
 20:    * Der Konstruktor reicht die an ihn &uuml;bergebenen Parameter an
 21:    * den Konstruktor der Klasse <CODE>StockItemImpl</CODE> weiter.
 22:    * Au&szlig;erdem speichert das Ausleihdatum ab.
 23:    */
 24:   public CassetteStoringStockItem(String name, Object date)
 25:   {
 26:     super (name);
 27:     this.date = date;
 28:   }
 29:
 30:
 31:   //// public methods ////////////////////////////////////////////////////
 32:
 33:   /**
 34:    * Gibt das Ausleihdatum der speziellen Videokassette zur&uuml;ck.
 35:    */
 36:   public Object getDate()
 37:   {
 38:     return date;
 39:   }
 40: }
```

```
 1: import sale.*;
 2: import sale.stdforms.*;
 3:
 4:
 5: /**
 6:  * Diese Klasse implementiert den eigentlichen "Verkaufsstand". Hier
 7:  * k&ouml;nnen Videos gekauft werden.
 8:  */
 9: public class Counter extends SalesPoint
10: {
11:
12:   //// constructor /////////////////////////////////////////////////////////
13:
14:   /**
15:    * Konstuktor.
16:    */
17:   public Counter(String name)
18:   {
19:     super(name);
20:   }
21:
22:
23:   //// public methods //////////////////////////////////////////////////////
24:
25:   /**
26:    * Gibt das Standard-FormSheet für diesen Counter zur&uuml;ck.
27:    */
28:   public FormSheet getDefaultFormSheet()
29:   {
30:     return new FormSheet("Menu", new DefaultCounterFormCreator(), false);
31:   }
32: }
```

```
  1: import sale.*;
  2: import data.*;
  3: import data.ooimpl.*;
  4: import data.events.*;
  5: import users.*;
  6:
  7:
  8: /**
  9:  * Ein registrierter Kunde mitsamt seinem Videobestand.
 10:  */
 11: public class Customer extends User
 12: {
 13:
 14:   //// attributes ////////////////////////////////////////////////////////
 15:
 16:   private String       customerID;               // Kundennummer
 17:   private StoringStock customerStoringStock;     // ausgeliehene Videos
 18:
 19:
 20:   //// constructor ///////////////////////////////////////////////////////
 21:
 22:   /**
 23:    * Legt einen neuen Kunden an.
 24:    */
 25:   public Customer(String customerID)
 26:   {
 27:     super(customerID);
 28:
 29:     this.customerID = customerID;
 30:     customerStoringStock = new StoringStockImpl(customerID,
 31:       (CatalogImpl)Shop.getTheShop().getCatalog("Video-Catalog"));
 32:     Shop.getTheShop().addStock(customerStoringStock);
 33:   }
 34:
 35:
 36:   //// public methods ////////////////////////////////////////////////////
 37:
 38:   /**
 39:    * Liefert die Kunden-ID.
 40:    */
 41:   public String getCustomerID()
 42:   {
 43:     return customerID;
 44:   }
 45:
 46:   /**
 47:    * F&uuml;gt die Kassette dem Bestand des Kunden hinzu.
 48:    */
 49:   public void addVideoCassette(CassetteStoringStockItem cassette)
 50:   {
 51:     customerStoringStock.add(cassette, null);
 52:   }
 53:
 54:   /**
 55:    * L&ouml;scht die Kassette aus dem Bestand des Kunden.
 56:    */
 57:   public void removeVideoCassette(CassetteStoringStockItem cassette)
 58:   {
 59:     try {
 60:       customerStoringStock.remove(cassette, null);
 61:     }
 62:     catch(VetoException ve) {
 63:     }
 64:   }
 65:
 66:   /**
 67:    * Liefert den gesamten Videobestand des Kunden.
 68:    */
 69:   public StoringStock getStoringStock()
 70:   {
 71:     return customerStoringStock;
 72:   }
 73:
 74: }
```

```
  1: import data.*;
  2: import sale.*;
  3: import util.swing.*;
  4:
  5: import java.awt.*;
  6: import java.text.*;
  7:
  8: import javax.swing.*;
  9:
 10:
 11: /**
 12:  * Editor f&uuml;r die Geldbetr&auml;ge in den Tabellen.
 13:  */
 14: public class DMCellEditor extends DefaultCellEditor
 15: {
 16:
 17:   //// attributes ////////////////////////////////////////////////////////
 18:
 19:   private String[] result;
 20:
 21:
 22:   //// constructor ///////////////////////////////////////////////////////
 23:
 24:   /**
 25:    * Erstellt ein neues Objekt der Klasse <CODE>DMCellEditor</CODE>.
 26:    */
 27:   public DMCellEditor (String[] result, String init)
 28:   {
 29:     super(new JTextInput(result, init));
 30:     this.result = result;
 31:   }
 32:
 33:
 34:   //// public methods ////////////////////////////////////////////////////
 35:
 36:   /**
 37:    * Gibt die zum Editieren verwendete Komponente korrekt initialisiert
 38:    * zur&uuml;ck.
 39:    */
 40:   public Component getTableCellEditorComponent(JTable jTable,
 41:     Object value, boolean isSelected, int row, int column)
 42:   {
 43:     // die Komponente holen,
 44:     Component component =
 45:       super.getTableCellEditorComponent(jTable, value, isSelected,
 46:                                         row, column);
 47:
 48:     // ...initialisieren
 49:     ((JTextInput)component).setText(((Currency)
 50:       Shop.getTheShop().getCatalog("DM")).toString((NumberValue)value));
 51:
 52:     // ...und zurueckgeben
 53:     return component;
 54:   }
 55:
 56:   /**
 57:    * Versucht, den eingegebenen Text als Geldbetrag zu interpretieren.
 58:    */
 59:   public Object getCellEditorValue()
 60:   {
 61:     // versuchen, den Text zu parsen und zurueckzugeben
 62:     try {
 63:       return ((Currency)Shop.getTheShop().getCatalog("DM")).parse(result[0]);
 64:     }
 65:     catch (ParseException pexc) {
 66:       // Text konnte nicht verarbeitet werden, 0 zurueckgeben
 67:       return new IntegerValue (0);
 68:     }
 69:   }
 70:
 71:   /**
 72:    * Versucht, das Editieren zu beenden und gibt eine Bewertung des
 73:    * Erfolges zur&uuml;ck.
 74:    */
 75:   public boolean stopCellEditing()
 76:   {
 77:     // versuchen den Tet zu parsen
 78:     try {
 79:       ((Currency)Shop.getTheShop().getCatalog("DM")).parse(result [0]);
 80:     }
 81:     catch (ParseException pexc) {
 82:       // Text konnte nicht verarbeitet werden, editieren kann nicht
 83:       // beendet werden
 84:       return false;
 85:     }
 86:
 87:     // Editieren kann nach Maszgabe der Methode der Oberklasse beendet werden
 88:     return super.stopCellEditing();
 89:   }
 90: }
```

```
  1: import sale.*;
  2: import data.ooimpl.*;
  3: import data.stdforms.*;
  4:
  5:
  6: /**
  7:  * Erzeugt eine Tabelle, sowie notwendige Buttons f&uuml;r die
  8:  * Angebotsanzeige.
  9:  */
 10: public class DefaultCounterFormCreator extends FormSheetContentCreator
 11: {
 12:
 13:   //// constructor ////////////////////////////////////////////////////////
 14:
 15:   /**
 16:    * Konstruktor.
 17:    */
 18:   public DefaultCounterFormCreator()
 19:   {
 20:     super();
 21:   }
 22:
 23:
 24:   //// public methods /////////////////////////////////////////////////////
 25:
 26:
 27:   /**
 28:    * Erzeugt den <CODE>FormSheetContent</CODE>, also Tabelle(n) und
 29:    * Buttons. Als Parameter wird das zu bearbeitende <CODE>FormSheet</CODE>
 30:    * &uuml;bergeben.
 31:    */
 32:   public void createFormSheetContent(FormSheet map)
 33:   {
 34:
 35:     // heraussuchen des Videobestandes zur Darstellung in der Tabelle
 36:     CountingStockImpl cs =
 37:       (CountingStockImpl)Shop.getTheShop().getStock("Video-Countingstock");
 38:
 39:     // erstellen des FormSheets mit der Tabelle des Bestandes als Inhalt
 40:     FormSheet fs = SingleTableFormSheet.create("Available Videocassettes",
 41:       cs, null, new OfferTED(false));
 42:
 43:     // die Tabelle holen und dem uebergebenen FormSheet als Komponente zuweisen
 44:     map.setComponent(fs.getComponent());
 45:
 46:     // alle Buttons entfernen
 47:     map.removeAllButtons();
 48:
 49:     map.addButton ("rent", 1,
 50:       new sale.Action()
 51:       {
 52:         public void doAction(SaleProcess p, SalesPoint s)
 53:         {
 54:           // Code zum Ausfuehren des Verleihvorgangs
 55:           s.runProcess(new RentProcess());
 56:         }
 57:       }
 58:     );
 59:
 60:     map.addButton ("give back", 2,
 61:       new sale.Action()
 62:       {
 63:         public void doAction(SaleProcess p, SalesPoint s)
 64:         {
 65:           //Code zum Ausfuehren des Rueckgabevorgangs
 66:           s.runProcess(new GiveBackProcess());
 67:         }
 68:       }
 69:     );
 70:   }
 71: }
```

```
  1: import data.*;
  2: import sale.*;
  3:
  4: import javax.swing.*;
  5:
  6: /**
  7:  * FormSheetContentCreator, der den Inhalt der Standard-FormSheets von
  8:  * Office erzeugt
  9:  */
 10: public class DefaultOfficeFormCreator extends FormSheetContentCreator
 11: {
 12:
 13:   //// attributes ////////////////////////////////////////////////////////
 14:
 15:   private Office officeOwner;
 16:
 17:
 18:   //// constructor ///////////////////////////////////////////////////////
 19:
 20:   /**
 21:    * Konstruktor. Erzeugt ein neues Objekt vom Typ
 22:    * <CODE>DefaultOfficeFormCreator</CODE>.
 23:    */
 24:   public DefaultOfficeFormCreator(Office officeOwner)
 25:   {
 26:     super();
 27:     this.officeOwner = officeOwner;
 28:   }
 29:
 30:
 31:   //// public methods ////////////////////////////////////////////////////
 32:
 33:   /**
 34:    * Erzeugt den Inhalt des &uuml;bergebenen FormSheets.
 35:    */
 36:   public void createFormSheetContent (FormSheet fsToCreate)
 37:   {
 38:     // JPanel erzeugen und vertikales BoxLayout setzen
 39:     JPanel jpFSComponent = new JPanel();
 40:
 41:     jpFSComponent.setLayout (new BoxLayout (jpFSComponent, BoxLayout.Y_AXIS));
 42:
 43:     // Label mit der aktuellen Zeit ins JPanel einfuegen
 44:     jpFSComponent.add(new JLabel("Turn : " +
 45:       Shop.getTheShop().getTimer().getTime()));
 46:
 47:     // Geldbestand holen
 48:     Stock coinSlot = Shop.getTheShop().getStock("coin slot");
 49:
 50:     // DataBasket des Managers holen
 51:     DataBasket db = officeOwner.getBasket();
 52:
 53:     // verwendete Waehrung holen
 54:     Currency currency = (Currency)Shop.getTheShop().getCatalog("DM");
 55:
 56:     // Geldbestand aufsummieren und Label mit der Summe ins JPanel einfuegen
 57:     IntegerValue money = (IntegerValue)coinSlot.sumStock(db,
 58:       new CatalogItemValue(), new IntegerValue (0));
 59:
 60:     jpFSComponent.add (new JLabel("Money : " + currency.toString(money)));
 61:
 62:     // Komponente ins FormSheet einfuegen
 63:     fsToCreate.setComponent (jpFSComponent);
 64:
 65:     // Alle vorhandenen Buttons entfernen
 66:     fsToCreate.removeAllButtons();
 67:
 68:      // Button zum Schliessen eibauen
 69:     fsToCreate.addButton ("Close", 1, new sale.Action() {
 70:       public void doAction (SaleProcess p, SalesPoint sp) {
 71:         officeOwner.quit();
 72:       }
 73:     });
 74:
 75:   }
 76: }
```

```
  1: import sale.*;
  2: import data.*;
  3: import data.ooimpl.*;
  4: import data.swing.*;
  5: import data.events.*;
  6: import util.swing.*;
  7:
  8: import java.util.*;
  9:
 10: import javax.swing.*;
 11: import javax.swing.table.*;
 12:
 13:
 14: /**
 15:   * Ein TableEntryDescriptor zur ausf&uuml;hrlichen Darstellung des
 16:   * Videobestandes.
 17:   */
 18: public class EditableVideoStockTED extends AbstractTableEntryDescriptor
 19: {
 20:
 21:   //// attributes ////////////////////////////////////////////////////////
 22:
 23:   private CountingStockImpl videoCountingStock;     // darzustellender Bestand
 24:   private DataBasket db;                            // verwendeter Datenkorb
 25:
 26:   // Variablen zur Formatierung der Darstellung
 27:   private TableCellRenderer tcrMoney;
 28:   private TableCellRenderer tcrName;
 29:   private TableCellRenderer tcrCount;
 30:
 31:   //// constructor ///////////////////////////////////////////////////////
 32:
 33:   /**
 34:    * Erzeugt ein neues Objekt der Klasse <CODE>EditableVideoStockTED</CODE>.
 35:    * Es wird der darzustellende <CODE>CountingStock</CODE> und der zu
 36:    * verwendende <CODE>DataBasket</CODE> &uuml;bergeben.
 37:    */
 38:   public EditableVideoStockTED(CountingStockImpl videoCountingStock,
 39:     DataBasket db)
 40:   {
 41:     super();
 42:     this.videoCountingStock = videoCountingStock;
 43:     this.db = db;
 44:
 45:     // Initialisierung der Variablen zu Darstellung der
 46:     // einzelnen Formate
 47:     tcrMoney =
 48:       new CurrencyRenderer((Currency)Shop.getTheShop().getCatalog("DM"));
 49:     tcrName  = new DefaultTableCellRenderer();
 50:     tcrCount = new DefaultTableCellRenderer();
 51:
 52:     // Ausrichtung fuer die Darstellung der Anzahl
 53:     ((DefaultTableCellRenderer)tcrCount).setHorizontalAlignment(
 54:       SwingConstants.CENTER);
 55:   }
 56:
 57:   //// public methods ////////////////////////////////////////////////////
 58:
 59:   /**
 60:    * Liefert die Anzahl der anzuzeigenden Spalten.
 61:    */
 62:   public int getColumnCount()
 63:   {
 64:     return 5;
 65:   }
 66:
 67:   /**
 68:    * Liefert die Spaltennamen.
 69:    */
 70:   public String getColumnName(int index)
 71:   {
 72:     return (new String[] {"Name",
 73:                           "Buy",
 74:                           "Sell",
 75:                           "Pieces in Stock",
 76:                           "Rent Pieces"})[index];
 77:   }
 78:
 79:   /**
 80:    * Legt die Darstellung f&uuml;r die einzelnen Spalten fest.
 81:    */
 82:   public TableCellRenderer getCellRenderer(int index)
 83:   {
 84:     switch (index) {
 85:       case 0: return tcrName;
 86:       case 3: return tcrCount;
 87:       case 4: return tcrCount;
 88:       default: return tcrMoney;
 89:     }
 90:   }
 91:
 92:   /**
 93:    * Wird aufgerufen, wenn <CODE>getCellRenderer()</CODE> oder
 94:    * <CODE>getCellEditor() null</CODE> zur&uuml;ckgeben.
 95:    */
 96:   public Class getColumnClass(int index)
 97:   {
 98:     return null;
 99:   }
100:
101:   /**
102:    * Liefert den Zelleninhalt f&uuml;r das &uuml;bergebene Objekt
103:    * und die angegebene Spalte.
104:    */
105:   public Object getValueAt(Object record, int index)
106:   {
107:     // Videokassette bestimmen
108:     VideoCassette videoCassette =
109:       (VideoCassette)((data.swing.CountingStockTableModel.Record)
110:                       record).getDescriptor();
111:     // Anzahl im Automaten ermitteln
112:     int count =
113:       ((data.swing.CountingStockTableModel.Record)record).getCount();
114:
115:     // Spalteneintrag zurueckgeben
116:     switch (index) {
117:       case 0:
118:         return videoCassette.getName();
119:       case 1:
120:         return ((QuoteValue)videoCassette.getValue()).getOffer();
121:       case 2:
122:         return ((QuoteValue)videoCassette.getValue()).getBid();
123:       case 3:
124:         return new IntegerValue(count);
125:       case 4:
126:         // Anzahl der ausgeliehenen Videos ermitteln
```

```
127:         int rented = 0;
128:         try {
129:           Iterator i = VideoMachine.getAllCustomer().iterator();
130:           while (i.hasNext()) {
131:             rented = rented +
132:               ((Customer)i.next()).getStoringStock().countItems(
133:                 videoCassette.getName(), null);
134:           }
135:         }
136:         catch (NullPointerException npe) {
137:         }
138:
139:         // und zurueckgeben
140:         return new IntegerValue(rented);
141:       default:
142:         return null;
143:     }
144:   }
145:
146:   /**
147:    * Definiert die Spalten, die editierbar sein sollen.
148:    */
149:   public boolean isElementEditable(Object record, int index)
150:   {
151:     return ((index == 1) || (index == 2));
152:   }
153:
154:
155:   /**
156:    * Liefert f&uuml;r die zu editierenden Spalten den
157:    * <CODE>TableCellEditor</CODE> zur&uuml;ck. In diesem Fall
158:    * eine Instanz der Klasse <CODE>DMCellEditor</CODE>.
159:    */
160:   public TableCellEditor getCellEditor(int index)
161:   {
162:     // Preise zu editieren?
163:     if (index == 1 || index == 2)
164:       return new DMCellEditor(new String[1], "");
165:
166:     // restliche Felder
167:     else
168:       return super.getCellEditor(index);
169:   }
170:
171:   /**
172:    * &Uuml;bertr&auml;gt den eingegebenen Wert in das jeweilige
173:    * Objekt.
174:    */
175:   public void setValueAt(Object record, int index, Object value)
176:   {
177:
178:     VideoCassette videoCassette =
179:       (VideoCassette)((data.swing.CountingStockTableModel.Record)
180:         record).getDescriptor();
181:
182:     try {
183:       VideoCassette videoCassetteToEdit =
184:         (VideoCassette)videoCountingStock.getCatalog(null).get(
185:         videoCassette.getName(), db, true);
186:       QuoteValue quoteValue = (QuoteValue)videoCassetteToEdit.getValue();
187:
188:       if (index == 1)
189:         videoCassetteToEdit.setValue(new QuoteValue(quoteValue.getBid(),
190:           (Value)value));
191:
192:       if (index == 2)
193:         videoCassetteToEdit.setValue(new QuoteValue((Value)value,
194:           quoteValue.getOffer()));
195:      }
196:
197:     catch (VetoException ve) {
198:       JOptionPane.showMessageDialog(null,
199:         "The editing of that food item was vetoed. It might be in use.");
200:
201:     }
202:   }
203:
204: }
```

```
  1: import sale.*;
  2: import sale.stdforms.*;
  3: import data.*;
  4: import data.ooimpl.*;
  5: import data.stdforms.*;
  6: import data.events.*;
  7: import users.*;
  8: import log.*;
  9:
 10: import java.util.*;
 11: import java.text.*;
 12: import java.lang.*;
 13: import java.io.*;
 14:
 15: import javax.swing.JTextField;
 16: import javax.swing.JPanel;
 17: import javax.swing.JOptionPane;
 18: import javax.swing.BoxLayout;
 19: import javax.swing.JLabel;
 20:
 21:
 22:
 23: /**
 24:  * R&uuml;ckgabeprocess, der von einem registrierten Kunden
 25:  * am Automaten durchgef&uuml;hrt werden kann.
 26:  */
 27: public class GiveBackProcess extends SaleProcess
 28: {
 29:
 30:   //// attributes ////////////////////////////////////////////////////////
 31:
 32:   // Gates
 33:   protected UIGate capabilityGate;
 34:   protected UIGate selectionGate;
 35:   protected UIGate giveRestGate;
 36:
 37:   // Transitions
 38:   protected Transition toSelectionTransition;
 39:   protected Transition toGetMoneyTransition;
 40:
 41:   // verwendete Waehrung
 42:   protected Currency myCurrency;
 43:
 44:   // rueckzuzahlender Betrag
 45:   private int toPayBackValue;
 46:
 47:   // gezahlter Betrag (Verkaufspreis des Videos)
 48:   private IntegerValue paidValue;
 49:
 50:   // Kunde, der zurueckgeben moechte
 51:   private Customer customer;
 52:
 53:   // Verleihpreis eines Videos pro Tag
 54:   private int rentPrice = 300;
 55:
 56:
 57:   //// constructor //////////////////////////////////////////////////////
 58:
 59:   /**
 60:    * Erzeugt ein neues Objekt der Klasse GiveBackProcess.
 61:    */
 62:   public GiveBackProcess()
 63:   {
 64:     super ("GiveBackProcess");
 65:   }
 66:
 67:   //// protected methods /////////////////////////////////////////////////
 68:
 69:   /**
 70:    * Baut die Oberfl&auml;che f&uuml;r den R&uuml;ckgabevorgang auf.
 71:    */
 72:   protected void setupMachine()
 73:   {
 74:     // verwendete Waehrung ermitteln
 75:     myCurrency = (Currency)Shop.getTheShop().getCatalog("DM");
 76:
 77:     // Videokatalog
 78:     final CatalogImpl videoCatalog =
 79:       (CatalogImpl)Shop.getTheShop().getCatalog("Video-Catalog");
 80:
 81:     // Gate zur Kundennummer-Abfrage anlegen
 82:     capabilityGate = new UIGate(null, null);
 83:
 84:     // Formsheet fuer Eingabe der Kundennummer
 85:     final TextInputForm tif = new TextInputForm("Customer-ID",
 86:                                                 "Customer-ID",
 87:                                                 "");
 88:
 89:     // Auswahl-Gate anlegen
 90:     selectionGate = new UIGate(null, null);
 91:
 92:     // zu verwendenden Datenkorb ermitteln
 93:     final DataBasket db = getBasket();
 94:
 95:     // Transition zum Selection Gate
 96:     toSelectionTransition = new Transition()
 97:     {
 98:       public Gate perform(SaleProcess pOwner, User usr)
 99:       {
100:         // zu verwendenden Bestand ermitteln
101:         StoringStock ss = customer.getStoringStock();
102:
103:         // am Gate darzustellendes FormSheet
104:         TwoTableFormSheet ttfs = TwoTableFormSheet.create(
105:           "Give Back",           // Titel des FormSheets
106:           ss,                    // Quell-StoringStock
107:           db,                    // Ziel-DataBasket
108:           selectionGate          // Gate, an dem das FormSheet darzustellen ist
109:         );
110:
111:         // FormSheetContentCreator am Auswahl-Gate anmelden
112:         ttfs.addContentCreator(new FormSheetContentCreator()
113:         {
114:           protected void createFormSheetContent(FormSheet fs)
115:           {
116:             // alle vorhandenen Buttons entfernen
117:             fs.removeAllButtons();
118:
119:             // neuen "OK"-Button einbauen
120:             fs.addButton ("Ok", 100, new sale.Action()
121:             {
122:               public void doAction (SaleProcess p, SalesPoint sp)
123:               {
124:                 // Transition zum Bezahlen als naechste Transition setzen
125:                 selectionGate.setNextTransition(toGetMoneyTransition);
126:               }
```

```
127:             });
128:
129:             // "Cancel"-Button einbauen
130:             fs.addButton ("Cancel", 101, new sale.Action()
131:             {
132:               public void doAction (SaleProcess p, SalesPoint sp)
133:               {
134:                 // Transition zum Rollback-Gate als naechste Transition setzen
135:                 selectionGate.setNextTransition(
136:                   GateChangeTransition.CHANGE_TO_ROLLBACK_GATE);
137:               }
138:             });
139:           }
140:         });
141:
142:         // erstelltes FormSheet am zu betretenden Gate setzen
143:         selectionGate.setFormSheet(ttfs);
144:
145:         // als naechstes zu betretendes Gate zurueckgeben
146:         return selectionGate;
147:       }
148:     };
149:
150:     // Gate zum Ausgeben des Restgeldes anlegen
151:     giveRestGate = new UIGate(null, null);
152:
153:     // Transition zum Gate, an dem das Restgeld gegeben wird
154:     toGetMoneyTransition = new Transition()
155:     {
156:       public Gate perform(SaleProcess pOwner, User usr)
157:       {
158:         // Parameter zum Aufsummieren des Datenkorbes festlegen
159:         DataBasketCondition dbc =
160:           DataBasketConditionImpl.ALL_STOCK_ITEMS;
161:         int sellValue = 0;
162:         int rentValue = 0;
163:
164:         // Videobestand des Automaten
165:         CountingStockImpl videoStock =
166:           (CountingStockImpl)Shop.getTheShop().getStock("Video-Countingstock");
167:
168:         // aktuelle Zeit ermitteln
169:         Object date = Shop.getTheShop().getTimer().getTime();
170:
171:         // rueckzuzahlenden Betrag ermitteln
172:         Iterator i = db.iterator(dbc);
173:         while (i.hasNext()) {
174:
175:           // spezielle Kassette ermitteln
176:           StoringStockItemDBEntry cassetteItem =
177:             (StoringStockItemDBEntry)i.next();
178:           CasseteStoringStockItem cassette =
179:             (CassetteStoringStockItem)cassetteItem.getValue();
180:
181:             // Verkaufspreis bestimmen
182:             try {
183:               sellValue =
184:                 ((NumberValue)((QuoteValue)(videoCatalog.get(
185:                 cassette.getName(), null, false).getValue()))).getBid()
186:                 ).getValue().intValue();
187:             }
188:             catch (VetoException ve) {}
189:
190:           // mind. einen Tag ausgeliehen?
191:           if ((((Long)date).intValue()
192:                - ((Long)cassette.getDate()).intValue()) > 0) {
193:
194:             // Verleihgebuehr bestimmen
195:             rentValue = (((Long)date).intValue()
196:               - ((Long)cassette.getDate()).intValue())
197:               * rentPrice;
198:
199:
200:             // Leihgebuehr kleiner als Verkaufspreis?
201:             if (rentValue < sellValue) {
202:               //rueckzuzahlenden Betrag erhoehen
203:               toPayBackValue = toPayBackValue + (sellValue - rentValue);
204:
205:               // Video in Automatenbestand zuruecklegen
206:               videoStock.add(cassette.getName(), 1, null);
207:
208:               // Vorgang in Logdatei eintragen
209:               try {
210:                 Log.getGlobalLog().log(new GiveBackLoggable(
211:                   cassetteItem.getSecondaryKey(),
212:                   customer.getCustomerID(), date));
213:               }
214:
215:               catch (LogNoOutputStreamException lnose) {}
216:               catch (IOException ioe) {}
217:
218:             }
219:
220:             // Leihgebuehr groesser als Verkaufspreis
221:             else {
222:               JOptionPane.showMessageDialog(null,
223:                 cassette.getName() + " is your's!");
224:             }
225:           }
226:
227:           // keinen Tag ausgeliehen (trotzdem 3,00 DM Leihgebuehr)
228:           if ((((Long)date).intValue()
229:             - ((Long)cassette.getDate()).intValue()) == 0) {
230:
231:             //rueckzuzahlenden Betrag erhoehen
232:             toPayBackValue = toPayBackValue + (sellValue - rentPrice);
233:
234:             // Video in Automatenbestand zuruecklegen
235:             videoStock.add(cassette.getName(), 1, null);
236:
237:             // Vorgang in Logdatei eintragen
238:             try {
239:               Log.getGlobalLog().log(new GiveBackLoggable(
240:                 cassetteItem.getSecondaryKey(),
241:                 customer.getCustomerID(), date));
242:               }
243:
244:               catch (LogNoOutputStreamException lnose) {}
245:               catch (IOException ioe) {}
246:
247:
248:           }
249:         }
250:
251:         // ueberpruefen, ob der Kunde noch Videos im Bestand hat
252:         if (customer.getStoringStock().size(null) == 0)
```

```
253:           VideoMachine.removeCustomer(customer);
254:
255:         // rueckzugebenden Betrag vom Geldbestand abziehen
256:         try {
257:             if (toPayBackValue > 0 ) {
258:                 ((CountingStock)Shop.getTheShop().getStock(
259:                   "coin slot")).remove("1-Pfennig-Stück",
260:                                        toPayBackValue,
261:                                        pOwner.getBasket());
262:             }
263:         }
264:         catch (VetoException ve) {}
265:
266:         // am Gate darzustellendes FormSheet
267:         MsgForm mf = new MsgForm(
268:           "Give Rest",
269:           "You get " +
270:           myCurrency.toString(new IntegerValue(toPayBackValue)) +
271:           " back");
272:
273:         // ContentCreator zur Neubelegung des "OK"-Buttons hinzufuegen
274:         mf.addContentCreator(new FormSheetContentCreator()
275:         {
276:           public void createFormSheetContent(FormSheet fs)
277:           {
278:             // neue Aktion setzen
279:             fs.getButton(FormSheet.BTNID_OK).setAction(new Action()
280:             {
281:               public void doAction(SaleProcess p, SalesPoint sp)
282:               {
283:                 // zum Commit-Gate fuehrende Transition als
284:                 // naechste Transition setzen
285:                 giveRestGate.setNextTransition(
286:                   GateChangeTransition.CHANGE_TO_COMMIT_GATE);
287:               }
288:             });
289:           }
290:         });
291:
292:         // erstelltes FormSheet am zu betretenden Gate setzen
293:         giveRestGate.setFormSheet(mf);
294:
295:         // als naechstes zu betretendes Gate zurueckgeben
296:         return giveRestGate;
297:       }
298:     };
299:
300:     // FormSheetContentCreator am Kundennummer-Eingabe-Gate anmelden
301:     tif.addContentCreator(new FormSheetContentCreator()
302:     {
303:       protected void createFormSheetContent(FormSheet fs)
304:       {
305:         // alle vorhandenen Buttons entfernen
306:         fs.removeAllButtons();
307:
308:         // neuen "OK"-Button einbauen
309:         fs.addButton ("Ok", 100, new sale.Action()
310:         {
311:           public void doAction (SaleProcess p, SalesPoint sp)
312:           {
313:             // Eingabe ueberpruefen
314:             String customerID = tif.getText();
315:             boolean isNotAnInteger = true;
316:
317:             try {
318:               new Integer(customerID);
319:               isNotAnInteger = false;
320:             }
321:             catch (NumberFormatException nfe) {}
322:
323:             // Eingabe korrekt -> selectionGate
324:             if (!isNotAnInteger && (new Integer(customerID)).intValue() > 0) {
325:
326:               // existiert der Kunde?
327:               try {
328:                 boolean customerExists = false;
329:                 Set customerSet = VideoMachine.getAllCustomer();
330:                 Iterator i = customerSet.iterator();
331:                 while (i.hasNext() && !customerExists){
332:                   Customer myCustomer = (Customer)i.next();
333:                   if (myCustomer.getCustomerID().equals(customerID)) {
334:                     customer = myCustomer;
335:                     customerExists = true;
336:                   }
337:                 }
338:
339:                 if (customerExists) {
340:                   capabilityGate.setNextTransition(
341:                     toSelectionTransition);
342:                 }
343:
344:                 else {
345:                   JOptionPane.showMessageDialog(null,
346:                     "This CustomerID doesn't exist!");
347:                   capabilityGate.setNextTransition(
348:                     new GateChangeTransition(capabilityGate));
349:                 }
350:               }
351:
352:               catch (NullPointerException ne) {
353:                 JOptionPane.showMessageDialog(null,
354:                   "This CustomerID doesn't exist!");
355:                 capabilityGate.setNextTransition(
356:                   new GateChangeTransition(capabilityGate));
357:               }
358:             }
359:
360:             // Eingabe falsch -> Kunden informieren und
361:             // Eingabe wiederholen
362:             else {
363:                 JOptionPane.showMessageDialog(null,
364:                   "CustomerID must be a positive number!");
365:                 capabilityGate.setNextTransition(
366:                   new GateChangeTransition(capabilityGate));
367:             }
368:           }
369:         });
370:
371:         // "Cancel"-Button einbauen
372:         fs.addButton ("Cancel", 101, new sale.Action()
373:         {
374:           public void doAction (SaleProcess p, SalesPoint sp)
375:           {
376:             // Transition zum Rollback-Gate als naechste Transition setzen
377:             capabilityGate.setNextTransition(
378:               GateChangeTransition.CHANGE_TO_ROLLBACK_GATE);
```

```
379:           }
380:         });
381:       }
382:     });
383:
384:     // FormSheet am entsprechenden Gate setzen
385:     capabilityGate.setFormSheet(tif);
386:
387:   }
388:
389:   /**
390:    * Beinhaltet den Logeintrag, der beim schlie&szlig;en des SalesPoints
391:    * aufgerufen wird.
392:    */
393:   protected void logSalesPointClosed()
394:   {
395:   }
396:
397:
398:   //// public methods ////////////////////////////////////////////////////
399:
400:   /**
401:    * Gibt das Startgate des Prozesses zur&uuml;ck.
402:    */
403:   public Gate getInitialGate()
404:   {
405:     // Automat aufbauen
406:     setupMachine();
407:
408:     // ...und Capability-Gate als Startgate zurueckgeben
409:     return capabilityGate;
410:   }
411:
412:   /**
413:    * &Uuml;bergibt das Log-Gate. Hier das Stop-Gate, da beim Beenden des
414:    * Prozesses kein Log-Eintrag geschrieben werden soll.
415:    */
416:   public Gate getLogGate()
417:   {
418:     return getStopGate();
419:   }
420:
421: }
422:
423:
424: /**
425:  * Definiert Die Schnittstelle <Code>Loggable</CODE>.
426:  */
427: class GiveBackLoggable implements Loggable
428: {
429:
430:   //// attributes ////////////////////////////////////////////////////////
431:
432:   String name;
433:   String customerID;
434:   Object date;
435:
436:
437:   //// constructor ///////////////////////////////////////////////////////
438:
439:   /**
440:    * Der Konstruktor legt ein neues Objekt der Klasse <CODE>MyLoggable</CODE>
441:    * an, wobei der Konstruktor von <CODE>Loggable</CODE> aufgerufen wird,
442:    * und die &uuml;bergebenen Variablen zugewiesen werden. Diese sind
443:    * wichtig f&uuml;r die Erstellung des Log-Eintrags.
444:    */
445:   public GiveBackLoggable(String name,
446:                           String customerID, Object date)
447:   {
448:     super();
449:     this.name = name;
450:     this.customerID = customerID;
451:     this.date = date;
452:   }
453:
454:
455:   public GiveBackLoggable(StoringStockItemDBEntry cassetteItem,
456:                           Customer customer, Object date)
457:   {
458:     super();
459:     name = cassetteItem.getSecondaryKey();
460:     this.customerID = customerID;
461:     this.date = date;
462:   }
463:
464:
465:   //// public methods ////////////////////////////////////////////////////
466:
467:   /**
468:    * Holt sich den Log-Eintrag, der aus den &uuml;bergebenen Daten
469:    * zusammengestellt wird. Dazu wird die extra implementierte
470:    * Klasse <CODE>MyLogEntry</CODE> genutzt.
471:    */
472:   public LogEntry getLogData()
473:   {
474:     return new GiveBackLogEntry(name, customerID, date);
475:   }
476: }
477:
478:
479:
480: /**
481:  * &Uuml;berschreibt die Klasse <Code>LogEntry</CODE> um einen
482:  * selbstdefinierten Log-Eintrag zu erm&ouml;glichen.
483:  */
484: class GiveBackLogEntry extends LogEntry
485: {
486:
487:   //// attributes ////////////////////////////////////////////////////////
488:
489:   String name;
490:   String customerID;
491:   Object date;
492:
493:
494:   //// constructor ///////////////////////////////////////////////////////
495:
496:   /**
497:    * Der Konstruktor legt ein neues Objekt der Klasse <CODE>MyLogEntry</CODE>
498:    * an, wobei der Konstruktor von <CODE>LogEntry</CODE> aufgerufen wird,
499:    * und die &uuml;bergebenen Variablen zugewiesen werden. Diese sind
500:    * wichtig f&uuml;r die Erstellung des Log-Eintrags.
501:    */
502:   public GiveBackLogEntry(String name, String customerID, Object date)
503:   {
504:     super();
```

```
505:     this.name = name;
506:     this.customerID = customerID;
507:     this.date = date;
508:   }
509:
510:
511:   //// public methods ////////////////////////////////////////////////////
512:
513:   /**
514:    * Spezifiziert das Aussehen des Log-Eintrags.
515:    */
516:   public String toString()
517:   {
518:     return name +
519:       " gave back by customer " + customerID +
520:       " (ID) at turn " + date;
521:   }
522: }
```

```
  1: import data.*;
  2: import data.swing.*;
  3:
  4: import sale.*;
  5:
  6:
  7: /**
  8:  * Ein TableEntryDescriptor zur Darstellung des Videoangebotes.
  9:  */
 10: public class OfferTED extends DefaultMoneyBagItemTED
 11: {
 12:
 13:   //// attributes ////////////////////////////////////////////////////////
 14:
 15:   private boolean withCount;  // Spalte fuer Anzahl mit anzeigen?
 16:
 17:
 18:   //// constructor ///////////////////////////////////////////////////////
 19:
 20:   /**
 21:    * Konstruktor. Erzeugt einen neues Objekt vom Typ OfferTED.
 22:    */
 23:   public OfferTED(boolean withCount)
 24:   {
 25:     super((Currency)Shop.getTheShop().getCatalog("DM"));
 26:     this.withCount = withCount;
 27:   }
 28:
 29:
 30:  //// public methods ////////////////////////////////////////////////////
 31:
 32:  /**
 33:    * Gibt die Spaltenanzahl der Tabelle zur&uuml;ck.
 34:    */
 35:   public int getColumnCount()
 36:   {
 37:     return withCount?3:2;
 38:   }
 39:
 40:   /**
 41:    * Gibt die &Uuml;berschrift einer Spalte zur&uuml;ck.
 42:    */
 43:   public String getColumnName(int nIndex)
 44:   {
 45:     return (new String[] {"Name", "Price", "Available"}) [nIndex];
 46:   }
 47:
 48:   /**
 49:    * Gibt den Wert einer Tabellenzelle zur&uuml;ck.
 50:    */
 51:   public Object getValueAt(Object oRecord, int nIndex)
 52:   {
 53:     // wenn Preis angefordert...
 54:     if (nIndex == 1) {
 55:       // die videokassette ermitteln
 56:       VideoCassette vidCassette = (VideoCassette)(
 57:         (CountingStockTableModel.Record)oRecord).getDescriptor();
 58:
 59:       // den Verkaufspreis ermitteln und zurueckgeben
 60:       return ((QuoteValue)vidCassette.getValue()).getBid();
 61:     }
 62:     else return super.getValueAt(oRecord, nIndex);
 63:   }
 64: }
```

```
  1: import sale.*;
  2: import sale.events.*;
  3: import sale.stdforms.*;
  4: import data.events.*;
  5: import data.ooimpl.*;
  6: import log.*;
  7: import log.stdforms.*;
  8:
  9: import java.io.*;
 10:
 11:
 12: /**
 13:  * Diese Klasse implementiert das B&uuml;ro. Hier kann der Manager das
 14:  * Logfile einsehen und Videos nachkaufen oder aus dem Bestand nehmen.
 15:  */
 16: public class Office extends SalesPoint
 17: {
 18:   //// attributes ////////////////////////////////////////////////////////
 19:
 20:   // Managerpasswort
 21:   private static String password;
 22:
 23:
 24:   //// constructor ///////////////////////////////////////////////////////
 25:
 26:   /**
 27:    * Konstuktor erzeugt ein neues Objekt der Klasse <CODE>Office</CODE>.
 28:    */
 29:   public Office(String name)
 30:   {
 31:     super(name);
 32:
 33:     // neuen StockChangeListener am Geldbestand des Automaten anmelden
 34:     ((MoneyBagImpl)Shop.getTheShop().getStock
 35:      ("coin slot")).addStockChangeListener(new StockChangeAdapter()
 36:      {
 37:        // StockItem wurde hinzugefuegt
 38:        public void commitAddStockItems (StockChangeEvent sce)
 39:        {
 40:          // hat sich der Geldbestand geaendert, FormSheet neu darstellen
 41:          if (getCurrentProcess() == null && hasUseableDisplay (null))
 42:            try {
 43:              setFormSheet (null, getDefaultFormSheet());
 44:            }
 45:            catch (InterruptedException iexc) {
 46:              System.err.println ("Update interrupted");
 47:            }
 48:        }
 49:
 50:        // StockItem wurde geloescht
 51:        public void commitRemoveStockItems (StockChangeEvent sce)
 52:        {
 53:          // hat sich der Geldbestand geaendert, FormSheet neu darstellen
 54:          if (getCurrentProcess() == null && hasUseableDisplay (null))
 55:            try {
 56:              setFormSheet (null, getDefaultFormSheet());
 57:            }
 58:            catch (InterruptedException iexc) {
 59:              System.err.println ("Update interrupted");
 60:            }
 61:        }
 62:      });
 63:
 64:     // neuen TimerListener am Timer des Automaten anmelden
 65:     Shop.getTheShop().getTimer().addTimerListener(new TimerAdapter()
 66:     {
 67:       public void onGoneAhead (TimerEvent trEvt)
 68:       {
 69:         // wurde Zeit weitergeschalten, FormSheet neu darstellen
 70:         if (hasUseableDisplay(null)) {
 71:           try {
 72:             setFormSheet(null, getDefaultFormSheet());
 73:           }
 74:           catch (InterruptedException ie) {}
 75:         }
 76:       }
 77:     });
 78:   }
 79:
 80:
 81:   //// public methods ////////////////////////////////////////////////////
 82:
 83:   /**
 84:    * Gibt das Standard-FormSheet für das B&uuml;ro zur&uuml;ck.
 85:    */
 86:   public FormSheet getDefaultFormSheet()
 87:   {
 88:     return new FormSheet("Office", new DefaultOfficeFormCreator(this), false);
 89:   }
 90:
 91:   /**
 92:     * Gibt das Standard-MenuSheet zurück.
 93:     */
 94:   public MenuSheet getDefaultMenuSheet()
 95:   {
 96:     // neues Menue anlegen, wird Wartungsmenue
 97:     MenuSheet msSubMenu = new MenuSheet("Maintenance");
 98:
 99:     // neuen Menuepunkt zum Weiterschalten der Zeit anlegen
100:     msSubMenu.add(new MenuSheetItem ("Advance time", new sale.Action()
101:     {
102:       public void doAction(SaleProcess p, SalesPoint sp)
103:       {
104:         // Zeit weiterschalten
105:         Shop.getTheShop().getTimer().goAhead();
106:       }
107:     }));
108:
109:     // neuen Menuepunkt zum Einsehen der Log-Datei anlegen
110:     msSubMenu.add (new MenuSheetItem ("See log file", new sale.Action()
111:     {
112:       public void doAction (SaleProcess p, SalesPoint sp)
113:       {
114:         try {
115:           // LogTableForm erzeugen
116:           FileInputStream fis = new FileInputStream ("machine.log");
117:           LogInputStream lis = new LogInputStream (fis);
118:           LogTableForm ltf = new LogTableForm ("View log file", lis);
119:
120:           // "Cancel"-Button entfernen
121:           ltf.removeButton (FormSheet.BTNID_CANCEL);
122:
123:           // FormSheet setzen
124:           setFormSheet (null, ltf);
125:         }
126:
```

```
127:         // falls Datei nicht existiert
128:         catch (FileNotFoundException fnfexc) {
129:           try {
130:             setFormSheet(null, new MsgForm("Error", "Log file not found."));
131:           }
132:           catch (InterruptedException iexc) {}
133:         }
134:
135:         // falls Fehler beim Lesen der Datei
136:         catch (IOException ioexc) {
137:           try {
138:             setFormSheet(null, new MsgForm("Error",
139:               "Log file corrupt. It might be empty."));
140:           }
141:           catch (InterruptedException iexc) {}
142:         }
143:
144:         // falls Setzen des FormSheets unterbrochen
145:         catch (InterruptedException iexc) {
146:           try {
147:             setFormSheet (null, new MsgForm ("Error", iexc.toString()));
148:           }
149:           catch (InterruptedException iexc2) {}
150:         }
151:       }
152:     }));
153:
154:
155:     // neuen Menuepunkt zum Einsehen und Editieren des Bestandes
156:     msSubMenu.add(new MenuSheetItem ("Edit Video-Stock", new sale.Action()
157:     {
158:       public void doAction(SaleProcess p, SalesPoint sp)
159:       {
160:         // Code zum Ausfuehren des Prozesses
161:           sp.runProcess(new SeeVideoStockProcess());
162:       }
163:     }));
164:
165:     // neues Menue anlegen, wird Menueleiste
166:     MenuSheet msMenu = new MenuSheet("office menu");
167:
168:     // Wartungsmenue in Menueleiste des Bueros einbauen
169:     msMenu.add(msSubMenu);
170:
171:     // Menüleiste zurückgeben
172:     return msMenu;
173:   }
174:
175:
176:
177:   /**
178:    * Setzt das Managerpasswort.
179:    */
180:   public static void setPassword(String password)
181:   {
182:     Office.password = password;
183:   }
184:
185:
186:   /**
187:    * Testet ob das &uuml;bergebene Passwort das Managerpasswort ist.
188:    */
189:   public static boolean testPassword(String password)
190:   {
191:     return password.equals(Office.password);
192:   }
193:
194: }
```

```
  1: import sale.*;
  2: import sale.stdforms.*;
  3: import data.*;
  4: import data.ooimpl.*;
  5: import data.stdforms.*;
  6: import users.*;
  7: import log.*;
  8:
  9: import java.util.*;
 10: import java.text.*;
 11: import java.lang.*;
 12: import java.io.*;
 13:
 14: import javax.swing.JTextField;
 15: import javax.swing.JPanel;
 16: import javax.swing.JOptionPane;
 17: import javax.swing.BoxLayout;
 18: import javax.swing.JLabel;
 19:
 20:
 21:
 22: /**
 23:  * Verkaufs- bzw. Verleihprozess, der von einem registrierten Kunden
 24:  * am Automaten durchgef&uuml;hrt werden kann.
 25:  */
 26: public class RentProcess extends SaleProcess
 27: {
 28:
 29:   //// attributes ////////////////////////////////////////////////////////
 30:
 31:   // Gates
 32:   protected UIGate capabilityGate;
 33:   protected UIGate selectionGate;
 34:   protected UIGate rentGate;
 35:   protected Gate   decisionGate;
 36:   protected UIGate getChangeGate;
 37:
 38:   // Transitions
 39:   protected Transition toSelectionTransition;
 40:   protected Transition toPayingTransition;
 41:   protected Transition toDecisionTransition;
 42:   protected Transition toGetChangeTransition;
 43:
 44:   // verwendete Waehrung
 45:   protected Currency myCurrency;
 46:
 47:   // zu zahlender Betrag
 48:   private IntegerValue toPayValue;
 49:
 50:   // gezahlter Betrag
 51:   private IntegerValue paidValue;
 52:
 53:   // Bewertung der Relation zwischen zu zahlendem und gezahltem Betrag
 54:   private int payAssessment;
 55:
 56:   // Kunde, der ausleihen moechte
 57:   private Customer customer;
 58:
 59:
 60:   //// constructor //////////////////////////////////////////////////////
 61:
 62:   /**
 63:    * Erzeugt ein neues Objekt der Klasse RentProcess.
 64:    */
 65:   public RentProcess()
 66:   {
 67:     super ("RentProcess");
 68:   }
 69:
 70:   //// protected methods /////////////////////////////////////////////////
 71:
 72:   /**
 73:    * Baut die Oberfl&auml;che f&uuml;r den Verleihvorgang auf.
 74:    */
 75:   protected void setupMachine()
 76:   {
 77:     // verwendete Waehrung ermitteln
 78:     myCurrency = (Currency)Shop.getTheShop().getCatalog("DM");
 79:
 80:     // Gate zur Kundennummer-Abfrage anlegen
 81:     capabilityGate = new UIGate(null, null);
 82:
 83:     // Formsheet fuer Eingabe der Kundennummer
 84:     final TextInputForm tif = new TextInputForm("Customer-ID",
 85:                                                 "Customer-ID",
 86:                                                 "");
 87:
 88:     // Auswahl-Gate anlegen
 89:     selectionGate = new UIGate(null, null);
 90:
 91:     // Gate zum Ausleihen/Bezahlen anlegen
 92:     rentGate = new UIGate(null, null);
 93:
 94:     // Gate zum Ausgeben des Wechselgeldes anlegen
 95:     getChangeGate = new UIGate(null, null);
 96:
 97:     // zu verwendenden Datenkorb ermitteln
 98:     final DataBasket db = getBasket();
 99:
100:     // zu verwendenden Bestand ermitteln
101:     final CountingStockImpl cs =
102:         (CountingStockImpl)Shop.getTheShop().getStock("Video-Countingstock");
103:
104:
105:     // Transition zum Selection Gate
106:     toSelectionTransition = new Transition()
107:     {
108:       public Gate perform(SaleProcess pOwner, User usr)
109:       {
110:         // am Gate darzustellendes FormSheet
111:         TwoTableFormSheet ttfs = TwoTableFormSheet.create(
112:           "Make your selection", // Titel des FormSheets
113:           cs,                    // Quell-CountingStock
114:           db,                    // Ziel-DataBasket
115:           selectionGate,         // Gate, an dem das FormSheet darzustellen ist
116:           null,                  // Comparator fuer Quelltabelle
117:           null,                  // Comparator furr Ziel-Tabelle
118:           false,        // Zeilen mit Anzahl == 0 in der Quelltabelle anzeigen?
119:           new OfferTED(true),    // TableEntryDescriptor furr Quelltabelle
120:           null,                  // TableEntryDescriptor furr Zieltabelle
121:           null                   // Transferstrategie
122:         );
123:
124:         // FormSheetContentCreator am Auswahl-Gate anmelden
125:         ttfs.addContentCreator(new FormSheetContentCreator()
126:         {
```

```
127:           protected void createFormSheetContent(FormSheet fs)
128:           {
129:             // alle vorhandenen Buttons entfernen
130:             fs.removeAllButtons();
131:
132:             // neuen "OK"-Button einbauen
133:             fs.addButton ("Ok", 100, new sale.Action()
134:             {
135:               public void doAction (SaleProcess p, SalesPoint sp)
136:               {
137:                 // Transition zum Bezahlen als naechste Transition setzen
138:                 selectionGate.setNextTransition(toPayingTransition);
139:               }
140:             });
141:
142:             // "Cancel"-Button einbauen
143:             fs.addButton ("Cancel", 101, new sale.Action()
144:             {
145:               public void doAction (SaleProcess p, SalesPoint sp)
146:               {
147:                 // Transition zum Rollback-Gate als naechste Transition setzen
148:                 selectionGate.setNextTransition(
149:                   GateChangeTransition.CHANGE_TO_ROLLBACK_GATE);
150:               }
151:             });
152:           }
153:         });
154:
155:         // erstelltes FormSheet am zu betretenden Gate setzen
156:         selectionGate.setFormSheet(ttfs);
157:
158:         // als naechstes zu betretendes Gate zurueckgeben
159:         return selectionGate;
160:       }
161:     };
162:
163:     // Transition zum Wechselgeld-Gate
164:     toGetChangeTransition = new Transition()
165:     {
166:       public Gate perform(SaleProcess pOwner, User usr)
167:       {
168:         // am Gate darzustellendes FormSheet
169:         MsgForm mf = new MsgForm(
170:           "Get Change",
171:           "You get " +
172:           myCurrency.toString((IntegerValue)paidValue.subtract(toPayValue)) +
173:           " Change.");
174:
175:         // ContentCreator zur Neubelegung des "OK"-Buttons hinzufuegen
176:         mf.addContentCreator(new FormSheetContentCreator()
177:         {
178:           public void createFormSheetContent(FormSheet fs)
179:           {
180:             // neue Aktion setzen
181:             fs.getButton(FormSheet.BTNID_OK).setAction(new Action()
182:             {
183:               public void doAction(SaleProcess p, SalesPoint sp)
184:               {
185:                 // zum Commit-Gate fuehrende Transition als
186:                 // naechste Transition setzen
187:                 getChangeGate.setNextTransition(
188:                   GateChangeTransition.CHANGE_TO_COMMIT_GATE);
189:               }
190:             });
191:           }
192:         });
193:
194:         // erstelltes FormSheet am zu betretenden Gate setzen
195:         getChangeGate.setFormSheet(mf);
196:
197:         // als naechstes zu betretendes Gate zurueckgeben
198:         return getChangeGate;
199:       }
200:     };
201:
202:     // Gate anlegen, das entscheidet, ob Wechselgeld gegeben, zum Bezahlen
203:     // oder zum Commit-Gate gegangen wird
204:     decisionGate = new Gate()
205:     {
206:       public Transition getNextTransition(SaleProcess pOwner, User usr)
207:         throws InterruptedException
208:       {
209:         switch(payAssessment) {
210:           // zuwenig bezahlt - Message anzeigen und zurueck zum Bezahlen
211:           case -1:
212:             FormSheet mf = new MsgForm("Error",
213:                                        "You have to pay more!",
214:                                        false);
215:             pOwner.getContext().popUpFormSheet(pOwner, mf);
216:             return new GateChangeTransition(rentGate);
217:
218:           // genau richtig bezahlt - zum Commit-Gate
219:           case  0:
220:             return GateChangeTransition.CHANGE_TO_COMMIT_GATE;
221:
222:           // zuviel bezahlt - zur Wechselgeldausgabe
223:           case  1:
224:             return toGetChangeTransition;
225:
226:           // falscher Wert in payAssessment - Fehlermeldung und Abbruch
227:           default:
228:             FormSheet mf2 = new MsgForm("Error",
229:                                         "Internal error at Decision Gate. " +
230:                                         "Will quit process.",
231:                                         false);
232:             pOwner.getContext().popUpFormSheet (pOwner, mf2);
233:             return GateChangeTransition.CHANGE_TO_QUIT_GATE;
234:         }
235:       }
236:     };
237:
238:     // Transition zum Entscheidungsgate
239:     toDecisionTransition = new Transition()
240:     {
241:       public Gate perform (SaleProcess pOwner, User usr)
242:       {
243:         // ausreichend Geld gegeben?
244:         if (paidValue.compareTo (toPayValue) >= 0) {
245:           // zu zahlenden Betrag im Safe des Restaurants ablegen, wenn
246:           // ueberhaupt etwas zu bezahlen ist
247:           if (toPayValue.getValue().intValue() > 0)
248:             ((CountingStock)Shop.getTheShop().getStock(
249:               "coin slot")).add("1-Pfennig-Stück",
250:                                 toPayValue.getValue().intValue(),
251:                                 pOwner.getBasket());
252:
```

```
253:           // Genau richtig gezahlt? - payAssessment entsprechend belegen
254:           if (paidValue.compareTo(toPayValue) == 0)
255:             payAssessment = 0;
256:           else
257:             payAssessment = 1;
258:         }
259:
260:         // nein, zuwenig bezahlt - payAssessment entsprechend belegen
261:         else
262:           payAssessment = -1;
263:
264:         // Entscheidungsgate als naechstes zu betretendes Gate zurueckgeben
265:         return decisionGate;
266:       }
267:     };
268:
269:     // Transition zum Gate, an dem zu bezahlen ist
270:     toPayingTransition = new Transition()
271:     {
272:       public Gate perform(SaleProcess pOwner, User usr)
273:       {
274:         // Parameter zum Aufsummieren des Datenkorbes festlegen
275:         final DataBasketCondition dbc =
276:             //DataBasketConditionImpl.ALL_STOCK_ITEMS;
277:           new DataBasketConditionImpl (DataBasketEntry.STOCK_ITEM_MAIN_KEY,
278:                                        null, cs, null, null);
279:         BasketEntryValue bev = BasketEntryValues.ONLY_STOCK_ITEMS;
280:         QuoteValue qvSum =
281:           new QuoteValue(new IntegerValue(0), new IntegerValue(0));
282:
283:         // ...und Datenkorb damit aufsummieren
284:         pOwner.getBasket().sumBasket (dbc, bev, qvSum);
285:
286:         // zu zahlenden Betrag ermitteln
287:         toPayValue = (IntegerValue)qvSum.getBid();
288:
289:         // FormSheet für das naechste Gate erstellen
290:         FormSheet tif =
291:           new TextInputForm("Paying",
292:                             "You have to pay " +
293:                             myCurrency.toString (toPayValue) + ".",
294:                             myCurrency.toString (toPayValue)
295:                             );
296:
297:         // FormSheetContentCreator zum Einbauen der Buttons anmelden
298:         tif.addContentCreator(new FormSheetContentCreator()
299:         {
300:           protected void createFormSheetContent(FormSheet fs)
301:           {
302:             // als 'final' markierte Version des FormSheets anlegen
303:             final TextInputForm tifFinal = (TextInputForm)fs;
304:
305:             // alle bisher vorhandenen Buttons entfernen
306:             fs.removeAllButtons();
307:
308:             // neuen "OK"-Button einbauen
309:             fs.addButton("Ok", 100, new sale.Action()
310:             {
311:               public void doAction (SaleProcess p, SalesPoint sp)
312:               {
313:                 try {
314:                   // eingegebenen Text verarbeiten
315:                   paidValue =
316:                     (IntegerValue)myCurrency.parse(tifFinal.getText());
317:
318:                   // Videos in den Bestand des Kunden uebernehmen
319:                   final Object date = Shop.getTheShop().getTimer().getTime();
320:                   Iterator i = db.iterator(dbc);
321:                   while (i.hasNext()) {
322:                     final CountingStockItemDBEntry cassetteItem =
323:                       (CountingStockItemDBEntry)i.next();
324:                     int number = cassetteItem.count();
325:                     for (; number > 0; number --) {
326:
327:                       // Vorgang in Logdatei eintragen
328:                       try {
329:                         Log.getGlobalLog().log(new MyLoggable(
330:                           cassetteItem.getSecondaryKey(),
331:                           customer.getCustomerID(), date));
332:                       }
333:
334:                       catch (LogNoOutputStreamException lnose) {}
335:                       catch (IOException ioe) {}
336:
337:                       // Kassetten in Bestande uebernehmen
338:                       customer.addVideoCassette(new CassetteStoringStockItem(
339:                         cassetteItem.getSecondaryKey(), date));
340:                     }
341:                   }
342:
343:                   // ...und naechste Transition setzen
344:                   rentGate.setNextTransition(toDecisionTransition);
345:                 }
346:                 catch(ParseException pexc) {
347:                   // eingegebener Text konnte nicht verarbeitet werden
348:                   // Meldung erzeugen und ausgeben
349:                   MsgForm mf = new MsgForm ("Error",
350:                                             "The specified amount does " +
351:                                             "not have an appropriate format.");
352:
353:                   try {
354:                     p.getContext().popUpFormSheet (p, mf);
355:                   }
356:                   catch(InterruptedException iexc) {
357:                   }
358:                 }
359:               }
360:             });
361:
362:             // "Back"-Button einbauen
363:             fs.addButton ("Back", 101, new sale.Action()
364:             {
365:               public void doAction (SaleProcess p, SalesPoint sp)
366:               {
367:                 // naechste Transition setzen, fuehrt ohne weitere Aktionen
368:                 // zum Auswahl-Gate
369:                 rentGate.setNextTransition(
370:                   new GateChangeTransition(selectionGate));
371:               }
372:             });
373:
374:             // "Cancel"-Button einbauen
375:             fs.addButton ("Cancel", 102, new sale.Action()
376:             {
377:               public void doAction (SaleProcess p, SalesPoint sp)
378:               {
```

```
379:                 // naechste Transition setzen, fuehrt zum Rollback-Gate
380:                 rentGate.setNextTransition(
381:                   GateChangeTransition.CHANGE_TO_ROLLBACK_GATE);
382:               }
383:             });
384:
385:           }
386:         });
387:
388:         // FormSheet am entsprechenden Gate setzen
389:         rentGate.setFormSheet(tif);
390:
391:         // ...und naechstes zu betretendes Gate zurueckgeben
392:         return rentGate;
393:       }
394:     };
395:
396:     // FormSheetContentCreator am Kundennummer-Eingabe-Gate anmelden
397:     tif.addContentCreator(new FormSheetContentCreator()
398:     {
399:       protected void createFormSheetContent(FormSheet fs)
400:       {
401:         // alle vorhandenen Buttons entfernen
402:         fs.removeAllButtons();
403:
404:         // neuen "OK"-Button einbauen
405:         fs.addButton ("Ok", 100, new sale.Action()
406:         {
407:           public void doAction (SaleProcess p, SalesPoint sp)
408:           {
409:             // Eingabe ueberpruefen
410:             String customerID = tif.getText();
411:             boolean isNotAnInteger = true;
412:
413:             try {
414:               new Integer(customerID);
415:               isNotAnInteger = false;
416:             }
417:             catch(NumberFormatException nfe) {
418:               isNotAnInteger = true;
419:             }
420:
421:             // Eingabe korrekt -> selectionGate
422:             if (!isNotAnInteger && (new Integer(customerID)).intValue() > 0) {
423:
424:               // existiert der Kunde?
425:               try {
426:                 customer = new Customer(customerID);
427:
428:                 // nein, dann neu anlegen und der Liste reg. Kunden hinzufuegen
429:                 VideoMachine.addCustomer(customer);
430:               }
431:
432:               // ja, dann diesen holen
433:               catch (DuplicateKeyException dke) {
434:                 customer = VideoMachine.getCustomerByID(customerID);
435:               }
436:
437:               capabilityGate.setNextTransition(toSelectionTransition);
438:
439:             }
440:
441:             // Eingabe falsch -> Kunden informieren und
442:             // Eingabe wiederholen
443:             else {
444:               JOptionPane.showMessageDialog(null,
445:                 "CustomerID must be a positive Number!");
446:               capabilityGate.setNextTransition(
447:                 new GateChangeTransition(capabilityGate));
448:             }
449:           }
450:         });
451:
452:         // "Cancel"-Button einbauen
453:         fs.addButton ("Cancel", 101, new sale.Action()
454:         {
455:           public void doAction (SaleProcess p, SalesPoint sp)
456:           {
457:             // Transition zum Rollback-Gate als naechste Transition setzen
458:             capabilityGate.setNextTransition(
459:               GateChangeTransition.CHANGE_TO_ROLLBACK_GATE);
460:           }
461:         });
462:       }
463:     });
464:
465:     // FormSheet am entsprechenden Gate setzen
466:     capabilityGate.setFormSheet(tif);
467:
468:   }
469:
470:
471:   //// public methods /////////////////////////////////////////////////////
472:
473:   /**
474:    * Gibt das Startgate des Prozesses zur&uuml;ck.
475:    */
476:   public Gate getInitialGate()
477:   {
478:     // Automat aufbauen
479:     setupMachine();
480:
481:     // ...und Capability-Gate als Startgate zurueckgeben
482:     return capabilityGate;
483:   }
484:
485:   /**
486:    * &Uuml;bergibt das Log-Gate. Hier das Stop-Gate, da beim Beenden des
487:    * Prozesses kein Log-Eintrag geschrieben werden soll.
488:    */
489:   public Gate getLogGate()
490:   {
491:     return getStopGate();
492:   }
493:
494: }
495:
496:
497:
498: /**
499:  * Definiert Die Schnittstelle <Code>Loggable</CODE>.
500:  */
501: class MyLoggable implements Loggable
502: {
503:
504:   //// attributes ////////////////////////////////////////////////////////
```

```
505:
506:   String name;
507:   String customerID;
508:   Object date;
509:
510:
511:   //// constructor /////////////////////////////////////////////////////////
512:
513:   /**
514:    * Der Konstruktor legt ein neues Objekt der Klasse <CODE>MyLoggable</CODE>
515:    * an, wobei der Konstruktor von <CODE>Loggable</CODE> aufgerufen wird,
516:    * und die &uuml;bergebenen Variablen zugewiesen werden. Diese sind
517:    * wichtig f&uuml;r die Erstellung des Log-Eintrags.
518:    */
519:   public MyLoggable(String name,
520:                     String customerID, Object date)
521:   {
522:     super();
523:     this.name = name;
524:     this.customerID = customerID;
525:     this.date = date;
526:   }
527:
528:
529:   //// public methods /////////////////////////////////////////////////////
530:
531:   /**
532:    * Holt sich den Log-Eintrag, der aus den &uuml;bergebenen Daten
533:    * zusammengestellt wird. Dazu wird die extra implementierte
534:    * Klasse <CODE>MyLogEntry</CODE> genutzt.
535:    */
536:   public LogEntry getLogData()
537:   {
538:     return new MyLogEntry(name, customerID, date);
539:   }
540: }
541:
542:
543:
544: /**
545:  * &Uuml;berschreibt die Klasse <Code>LogEntry</CODE> um einen
546:  * selbstdefinierten Log-Eintrag zu erm&ouml;glichen.
547:  */
548: class MyLogEntry extends LogEntry
549: {
550:
551:   //// attributes ////////////////////////////////////////////////////////
552:
553:   String name;
554:   String customerID;
555:   Object date;
556:
557:
558:   //// constructor /////////////////////////////////////////////////////////
559:
560:   /**
561:    * Der Konstruktor legt ein neues Objekt der Klasse <CODE>MyLogEntry</CODE>
562:    * an, wobei der Konstruktor von <CODE>LogEntry</CODE> aufgerufen wird,
563:    * und die &uuml;bergebenen Variablen zugewiesen werden. Diese sind
564:    * wichtig f&uuml;r die Erstellung des Log-Eintrags.
565:    */
566:   public MyLogEntry(String name, String customerID, Object date)
567:   {
568:     super();
569:     this.name = name;
570:     this.customerID = customerID;
571:     this.date = date;
572:   }
573:
574:
575:   //// public methods /////////////////////////////////////////////////////
576:
577:   /**
578:    * Spezifiziert das Aussehen des Log-Eintrags.
579:    */
580:   public String toString()
581:   {
582:     return name +
583:       " rent by customer " + customerID +
584:       " (ID) at turn " + date;
585:   }
586: }
```

```
  1: import sale.*;
  2: import data.*;
  3: import data.ooimpl.*;
  4: import data.stdforms.*;
  5: import data.swing.*;
  6: import data.events.*;
  7: import users.*;
  8:
  9: import java.util.*;
 10:
 11: import javax.swing.*;
 12:
 13:
 14: /**
 15:  * Prozess zur Bestandsanzeige und zum Editieren des Bestandes.
 16:  */
 17: public class SeeVideoStockProcess extends SaleProcess
 18: {
 19:
 20:   //// attributes ////////////////////////////////////////////////////////
 21:
 22:   // Gates
 23:   protected UIGate selectionGate;
 24:
 25:   // Transitions
 26:   protected Transition newVideoTransition;
 27:   protected Transition removeVideoTransition;
 28:
 29:
 30:   //// constructor //////////////////////////////////////////////////////
 31:
 32:   /**
 33:    * Erzeugt ein neues Objekt der Klasse <CODE>SeeVideoStockProcess</CODE>.
 34:    */
 35:   public SeeVideoStockProcess()
 36:   {
 37:     super ("SeeVideoProcess");
 38:   }
 39:
 40:   //// protected methods /////////////////////////////////////////////////
 41:
 42:   /**
 43:    * Baut die Oberfl&auml;che f&uuml;r den Verleihvorgang auf.
 44:    */
 45:   protected void setupMachine()
 46:   {
 47:     // Auswahl-Gate anlegen
 48:     selectionGate = new UIGate(null, null);
 49:
 50:     // darzustellendes Formsheet erstellen
 51:     final SingleTableFormSheet stfs =
 52:       SingleTableFormSheet.create("See videos in stock and edit Process",
 53:         (CountingStockImpl)Shop.getTheShop().getStock("Video-Countingstock"),
 54:         selectionGate,
 55:         getBasket(),
 56:         true,
 57:         new EditableVideoStockTED((CountingStockImpl)
 58:           Shop.getTheShop().getStock("Video-Countingstock"), getBasket()));
 59:
 60:
 61:     // Transition zum Anlegen eines neuen Videos
 62:     newVideoTransition = new Transition()
 63:     {
 64:       public Gate perform(SaleProcess pOwner, User usr)
 65:       {
 66:         // Anlegen eines neuen Videos
 67:         // Eingabefelder erzeugen
 68:         JTextField jTextField1 = new JTextField();
 69:         JTextField jTextField2 = new JTextField();
 70:         JTextField jTextField3 = new JTextField();
 71:         JTextField jTextField4 = new JTextField();
 72:
 73:
 74:         // neues JPanel anlegen und vertikales Boxlayout setzen
 75:         JPanel jTextPanel = new JPanel();
 76:         jTextPanel.setLayout(new BoxLayout(jTextPanel,
 77:                                            BoxLayout.Y_AXIS));
 78:
 79:         // Label und Paßwortfelder einfuegen
 80:         jTextPanel.add(new JLabel("Name"));
 81:         jTextPanel.add(jTextField1);
 82:         jTextPanel.add(new JLabel("Buy (in Pf)"));
 83:         jTextPanel.add(jTextField2);
 84:         jTextPanel.add(new JLabel("Sell (in Pf)"));
 85:         jTextPanel.add(jTextField3);
 86:         jTextPanel.add(new JLabel("Amount"));
 87:         jTextPanel.add(jTextField4);
 88:
 89:         // Dialog darstellen
 90:         JOptionPane.showMessageDialog (null,
 91:                                        jTextPanel,
 92:                                        "New Video-Cassette",
 93:                                        JOptionPane.QUESTION_MESSAGE);
 94:
 95:         // Video-Katalog holen
 96:         Catalog videoCatalog =
 97:           Shop.getTheShop().getCatalog("Video-Catalog");
 98:
 99:         // Video-Bestand holen
100:         CountingStock videoCountingStock =
101:           (CountingStock)Shop.getTheShop().getStock("Video-Countingstock");
102:
103:         // Name des Videos
104:         String name = jTextField1.getText();
105:
106:         // Verkaufspreis
107:         int buy = 0;
108:         try {
109:           buy = (new Integer(jTextField2.getText())).intValue();
110:         }
111:         catch (NumberFormatException ne) {}
112:
113:         // Einkaufspreis
114:         int sell = 0;
115:         try {
116:           sell = (new Integer(jTextField3.getText())).intValue();
117:         }
118:         catch (NumberFormatException ne) {}
119:
120:         // Anzahl
121:         int amount = 0;
122:         try {
123:           amount = (new Integer(jTextField4.getText())).intValue();
124:         }
125:         catch (NumberFormatException ne) {}
126:
```

```
127:         // ueberpruefen, ob Eingabe korrekt
128:         if (sell > 0 && buy > 0 && amount > 0 && !name.equals("")) {
129:           // in Katalog uebernehmen
130:           try {
131:             videoCatalog.add(new VideoCassette(name, new QuoteValue
132:               (new IntegerValue (buy),
133:                new IntegerValue (sell))), null);
134:
135:             // in Bestand uebernehmen
136:             videoCountingStock.add(name, amount, null);
137:           }
138:
139:           // Element existiert bereits
140:           catch (DuplicateKeyException dke) {
141:             JOptionPane.showMessageDialog(null,
142:             "Element exists");
143:           }
144:
145:         }
146:
147:         // Eingabe fehlerhaft
148:         else {
149:           JOptionPane.showMessageDialog(null,
150:             "The given input was not correct!");
151:         }
152:
153:         // Auswahl-Gate zurueckgeben
154:         return selectionGate;
155:       }
156:     };
157:
158:     // Transition zum Loeschen anlegen
159:     removeVideoTransition = new Transition()
160:     {
161:       public Gate perform(SaleProcess pOwner, User usr)
162:       {
163:         // markierte Zeile holen
164:         Object record = stfs.getSelectedRecord();
165:         VideoCassette videoCassette =
166:           (VideoCassette)((data.swing.CountingStockTableModel.Record)
167:           record).getDescriptor();
168:
169:         // Anzahl der entliehenen Videos bestimmen
170:         int rented = 0;
171:         try {
172:           Iterator i = VideoMachine.getAllCustomer().iterator();
173:           while (i.hasNext()) {
174:             rented = rented +
175:               ((Customer)i.next()).getStoringStock().countItems(
176:                 videoCassette.getName(), null);
177:           }
178:         }
179:         catch (NullPointerException npe) {
180:         }
181:
182:
183:         // ueberpruefen, ob gewaehltes Video ausgeliehen
184:         if (rented <= 0) {
185:           // loeschen des gewaehlten Eintrags
186:           try {
187:             // DataBasket db = new DataBasketImpl();
188:             Shop.getTheShop().getCatalog("Video-Catalog").remove(
189:               videoCassette, null);
190:             // db.commit();
191:           }
192:           catch (VetoException ve) {
193:             ve.printStackTrace();
194:             JOptionPane.showMessageDialog(null,
195:               "The selected item can't be removed!");
196:           }
197:         }
198:
199:         // ausgeliehen Videos vorhanden
200:         else {
201:           JOptionPane.showMessageDialog(null,
202:             "There are still rented videos!");
203:         }
204:
205:         return selectionGate;
206:       }
207:     };
208:
209:
210:     // FormSheetContentCreator am Auswahl-Gate anmelden
211:     stfs.addContentCreator(new FormSheetContentCreator()
212:     {
213:       protected void createFormSheetContent(FormSheet fs)
214:       {
215:         // alle vorhandenen Buttons entfernen
216:         fs.removeAllButtons();
217:
218:         // neuen "New"-Button einbauen
219:         fs.addButton ("New", 100, new sale.Action()
220:         {
221:           public void doAction (SaleProcess p, SalesPoint sp)
222:           {
223:             // Transition zum Erstellen eines neuen Videos als
224:             // naechste Transition setzen
225:             selectionGate.setNextTransition(newVideoTransition);
226:           }
227:         });
228:
229:         // neuen "Remove"-Button einbauen
230:         fs.addButton ("Remove", 101, new sale.Action()
231:         {
232:           public void doAction (SaleProcess p, SalesPoint sp)
233:           {
234:             // Transition zum Loeschen eines Videos als naechste Transition
235:             // setzen
236:             selectionGate.setNextTransition(removeVideoTransition);
237:           }
238:         });
239:
240:         // neuen "Ok"-Button einbauen
241:         fs.addButton ("Ok", 102, new sale.Action()
242:         {
243:           public void doAction (SaleProcess p, SalesPoint sp)
244:           {
245:             // Transition zum Commit-Gate als naechste Transition setzen
246:             selectionGate.setNextTransition(
247:               GateChangeTransition.CHANGE_TO_COMMIT_GATE);
248:           }
249:         });
250:
251:         // neuen "Cancel"-Button einbauen
252:         fs.addButton ("Cancel", 103, new sale.Action()
```

```
253:         {
254:           public void doAction (SaleProcess p, SalesPoint sp)
255:           {
256:             // Transition zum Rollback-Gate als naechste Transition setzen
257:             selectionGate.setNextTransition(
258:               GateChangeTransition.CHANGE_TO_ROLLBACK_GATE);
259:           }
260:         });
261:
262:       }
263:     });
264:   }
265:
266:
267:   //// public methods ////////////////////////////////////////////////////
268:
269:   /**
270:    * Gibt das Startgate des Prozesses zur&uuml;ck.
271:    */
272:   public Gate getInitialGate()
273:   {
274:     // Automat aufbauen
275:     setupMachine();
276:
277:     // ...und Auswahl-Gate als Startgate zurueckgeben
278:     return selectionGate;
279:   }
280:
281:   /**
282:    * &Uuml;bergibt das Log-Gate. Hier das Stop-Gate, da beim Beenden des
283:    * Prozesses kein Log-Eintrag geschrieben werden soll.
284:    */
285:   public Gate getLogGate()
286:   {
287:     return getStopGate();
288:   }
289:
290: }
```

```
  1: import data.ooimpl.*;
  2: import data.*;
  3:
  4:
  5: public class VideoCassette extends CatalogItemImpl
  6: {
  7:
  8:   //// constructor /////////////////////////////////////////////////////////
  9:
 10:   /**
 11:    * Der Konstruktor reicht die an ihn &uuml;bergebenen Parameter an
 12:    * den Konstruktor der Klasse <CODE>CatalogItemImpl</CODE> weiter.
 13:    */
 14:   public VideoCassette (String name, QuoteValue value)
 15:   {
 16:     super (name, value);
 17:   }
 18:
 19:
 20:   //// public methods /////////////////////////////////////////////////////
 21:
 22:   /**
 23:    * Setzt den Wert der speziellen Videokassette.
 24:    */
 25:   public void setValue (QuoteValue value)
 26:   {
 27:     super.setValue (value);
 28:   }
 29:
 30:
 31:   //// protected methods //////////////////////////////////////////////////
 32:
 33:   /**
 34:    * Erzeugt eine Kopie des Objektes der Klasse <CODE>VideoCassette</CODE>
 35:    * mit gleichem Namen und gleichem Wert.
 36:    */
 37:   protected CatalogItemImpl getShallowClone()
 38:   {
 39:     return new VideoCassette (new String(getName()),
 40:       (QuoteValue)getValue().clone());
 41:   }
 42:
 43: }
```

```
  1: import sale.*;
  2: import log.*;
  3: import users.*;
  4: import data.*;
  5: import data.ooimpl.*;
  6:
  7: import java.io.*;
  8: import java.util.*;
  9: import java.lang.*;
 10:
 11: import javax.swing.*;
 12:
 13:
 14: /**
 15:  * Diese Klasse ist der "Shop" der Anwendung. Sie stellt das Grundger&uuml;st
 16:  * f&uuml;r die Anwendung dar.
 17:  */
 18: public class VideoMachine extends Shop
 19: {
 20:
 21:   //// attributes ////////////////////////////////////////////////////////////
 22:
 23:   // Liste aller registrierten Kunden
 24:   private static Set customerSet = new HashSet();
 25:   //// constructor ///////////////////////////////////////////////////////////
 26:
 27:   /**
 28:    * Konstruktor. Erzeugt ein neues Objekt vom Typ VideoAutomat.
 29:    */
 30:   public VideoMachine()
 31:   {
 32:     super();
 33:
 34:     // Name und Preis fuer die Standart-Katalogeintraege
 35:     String[] videos = {"Video 01", "Video 02", "Video 03", "Video 04",
 36:                        "Video 05", "Video 06", "Video 07", "Video 08",
 37:                        "Video 09", "Video 10"};
 38:     int[] buy  = {5000, 5000, 5000, 5000, 5000, 4000, 4000, 4000, 3000, 3000};
 39:     int[] sell = {4000, 4000, 4000, 4000, 3500, 3500, 3500, 3500, 3000, 3000};
 40:
 41:
 42:     // Eintragung in den Catalog und den Stock
 43:     Catalog videoCatalog = new CatalogImpl("Video-Catalog");
 44:     addCatalog(videoCatalog);
 45:
 46:     for (int i = 0; i < videos.length; i++) {
 47:       videoCatalog.add(new VideoCassette (videos[i], new QuoteValue
 48:         (new IntegerValue (buy[i]), new IntegerValue (sell[i]))), null);
 49:     }
 50:
 51:     CountingStock cs = new CountingStockImpl("Video-Countingstock",
 52:       (CatalogImpl)videoCatalog);
 53:     addStock(cs);
 54:
 55:     // Erstellen eines Bestands von je 5 Videos (sollte spaeter
 56:     // vom Manager uebernommen werden
 57:     Iterator cassettes = videoCatalog.keySet(null).iterator();
 58:     while (cassettes.hasNext()) {
 59:       cs.add((String)cassettes.next(), 5, null);
 60:     }
 61:
 62:     // Anlegen einer Waehrung und Erstellen eines dazugehoerigen
 63:     // Geldbestandes
 64:     addCatalog(new CurrencyImpl("DM"));
 65:
 66:     MoneyBag coinSlot = new MoneyBagImpl("coin slot",
 67:       (CurrencyImpl)getCatalog("DM"));
 68:     coinSlot.add("1-Pfennig-Stück", 100000, null);
 69:     addStock(coinSlot);
 70:   }
 71:
 72:
 73:   //// public methods ////////////////////////////////////////////////////////
 74:
 75:   /**
 76:    * Die Main-Methode startet die Anwendung.
 77:    */
 78:   public static void main (String[] args)
 79:   {
 80:     // neuen Videoautomaten erzeugen
 81:     VideoMachine vidMachine = new VideoMachine();
 82:     setTheShop(vidMachine);
 83:
 84:     // Log umdefinieren...
 85:     //    Log.setLogCreator(new MyLogCreator());
 86:
 87:     // Datei zuweisen, in die das Log geschrieben werden soll
 88:     try {
 89:       Log.setGlobalOutputStream(new FileOutputStream("machine.log", true));
 90:     }
 91:     catch (IOException ioex) {
 92:       System.err.println("Unable to create log file.");
 93:     }
 94:
 95:     // Verleih bzw. Kasse anlegen
 96:     Counter c = new Counter("Video Rental");
 97:     c.attach(new DataBasketImpl());
 98:     c.attach(new User("SalespointUser"));
 99:     vidMachine.addSalesPoint(c);
100:
101:     // Anlegen des Manager-Passworts
102:     // Paßwortfelder erzeugen
103:     JPasswordField jPasswordField1 = new JPasswordField();
104:     JPasswordField jPasswordField2 = new JPasswordField();
105:
106:     // neues JPanel anlegen und vertikales Boxlayout setzen
107:     JPanel jPasswordRequest = new JPanel();
108:     jPasswordRequest.setLayout(new BoxLayout(jPasswordRequest,
109:                                              BoxLayout.Y_AXIS));
110:
111:     // Label und Paßwortfelder einfuegen
112:     jPasswordRequest.add(new JLabel("Enter the manger's password:"));
113:     jPasswordRequest.add(jPasswordField1);
114:     jPasswordRequest.add(new JLabel("Re-enter the manager's password:"));
115:     jPasswordRequest.add(jPasswordField2);
116:
117:     // Paßwortabfrage wiederholen bis Uebereinstimmung zwischen den Worten
118:     do {
119:       JOptionPane.showMessageDialog (null,
120:                                      jPasswordRequest,
121:                                      "Manager",
122:                                      JOptionPane.QUESTION_MESSAGE);
123:     } while (!new String(jPasswordField1.getPassword()).equals(
124:         new String(jPasswordField2.getPassword())));
125:
126:     // Managerpasswort setzen
```

```
127:     Office.setPassword(new String(jPasswordField1.getPassword()));
128:
129:     // Titel setzen und starten
130:     vidMachine.setShopFrameTitle("Videoverleihautomat *** HOMECINEMA *** 24h");
131:     vidMachine.start();
132:
133:     vidMachine.getShopFrame().setSize(640,480);
134:     vidMachine.getShopFrame().validate();
135:   }
136:
137:
138:   /**
139:    * Erzeugt die Menueleiste des Hauptfensters unseres Automatens.
140:    */
141:   public MenuSheet createShopMenuSheet()
142:   {
143:     // urspruengliche Menueleiste erzeugen und abspeichern
144:     MenuSheet msMenuBar = super.createShopMenuSheet();
145:
146:     // 'Shop'-Menue heraussuchen lassen
147:     MenuSheet msShopMenu =
148:       (MenuSheet)msMenuBar.getTaggedItem (SHOP_MENU_TAG, true);
149:
150:     // nach unten zu schiebende Menuepunkte zunaechst aus dem
151:     // Menue loeschen, aber in lokalen Variablen speichern
152:     MenuSheetItem msLoadItem = (MenuSheetItem)msShopMenu.remove(LOAD_TAG);
153:     MenuSheetItem msSaveItem = (MenuSheetItem)msShopMenu.remove(SAVE_TAG);
154:     MenuSheetSeparator msSeparator =
155:       (MenuSheetSeparator)msShopMenu.remove(SEPARATOR_TWO_TAG);
156:     MenuSheetItem msQuitItem = (MenuSheetItem)msShopMenu.remove(QUIT_SHOP_TAG);
157:
158:     MenuSheetItem msManagerItem = new MenuSheetItem("Open Office",
159:       new sale.Action()
160:       {
161:         public void doAction (SaleProcess p, SalesPoint sp)
162:         {
163:           List l = getSalesPoints();
164:           for (int i = 0; i < l.size(); i++) {
165:             if (((SalesPoint)l.get(i)).getName() == "Office") {
166:               javax.swing.JOptionPane.showMessageDialog(null,
167:                 "There is an already an Office",
168:                 "Error",
169:                 javax.swing.JOptionPane.ERROR_MESSAGE);
170:               return;
171:             }
172:           }
173:
174:           // Paßwortfeld erzeugen
175:           JPasswordField jPasswordField = new JPasswordField();
176:
177:           // neues JPanel anlegen und vertikales Boxlayout setzen
178:           JPanel jPasswordRequest = new JPanel();
179:           jPasswordRequest.setLayout(new BoxLayout(jPasswordRequest,
180:                                                    BoxLayout.Y_AXIS));
181:
182:           // Label und Passwortfelder einfuegen
183:           jPasswordRequest.add(new JLabel("Enter the manger's password:"));
184:           jPasswordRequest.add(jPasswordField);
185:
186:           // Passwortabfrage
187:           JOptionPane.showMessageDialog (null,
188:                                          jPasswordRequest,
189:                                          "Manager",
190:                                          JOptionPane.QUESTION_MESSAGE);
191:
192:           // ist eingegebenes Passwort korrekt?
193:           if (Office.testPassword(new String(jPasswordField.getPassword()))) {
194:
195:             // ja -> Buero anlegen
196:             Office o = new Office("Office");
197:             o.attach (new DataBasketImpl());
198:             o.attach (new User("Manager"));
199:             Shop.getTheShop().addSalesPoint (o);
200:           }
201:
202:           // nein -> Info an Nutzer
203:           else {
204:             JOptionPane.showMessageDialog(null,
205:                 "Access denied!");
206:           }
207:         }
208:         });
209:
210:     // Menueeintraege in der gewuenschten Reihenfolge wieder einfuegen
211:     msShopMenu.add (msManagerItem);
212:     msShopMenu.add (msSeparator);
213:     msShopMenu.add (msLoadItem);
214:     msShopMenu.add (msSaveItem);
215:     msShopMenu.add (msSeparator);
216:     msShopMenu.add (msQuitItem);
217:
218:     // Menueleiste zurueckgeben
219:     return msMenuBar;
220:   }
221:
222:
223:   /**
224:     * Beendet das Programm ohne den Stand abzuspeichern.
225:     */
226:   public void quit()
227:   {
228:     if (Shop.getTheShop().shutdown (false)) {
229:       System.exit (0);
230:     }
231:   }
232:
233:   /**
234:    * F&uuml;gt der Liste registrierter Kunden einen neuen hinzu.
235:    */
236:   public static void addCustomer(Customer customer)
237:   {
238:     customerSet.add(customer);
239:   }
240:
241:   /**
242:    * L&ouml;scht den angegebenen Kunden aus der Liste registrierter
243:    * Kunden heraus.
244:    */
245:   public static void removeCustomer(Customer customer)
246:   {
247:     customerSet.remove(customer);
248:   }
249:
250:   /**
251:    * Liefert eine Liste aller registrierter Kunden.
252:    */
```

```
253:   public static Set getAllCustomer()
254:   {
255:     return customerSet.isEmpty() ? null : customerSet;
256:   }
257:
258:
259:   /**
260:    * Liefert das Kundenobject zur &uuml;bergebenen Kundennummer.
261:    */
262:   public static Customer getCustomerByID(String customerID)
263:   {
264:     Iterator i = customerSet.iterator();
265:     while (i.hasNext()) {
266:       Customer customer = (Customer)i.next();
267:       if (customer.getCustomerID().equals(customerID))
268:         return customer;
269:     }
270:     return null;
271:   }
272: }
```

Table of Contents