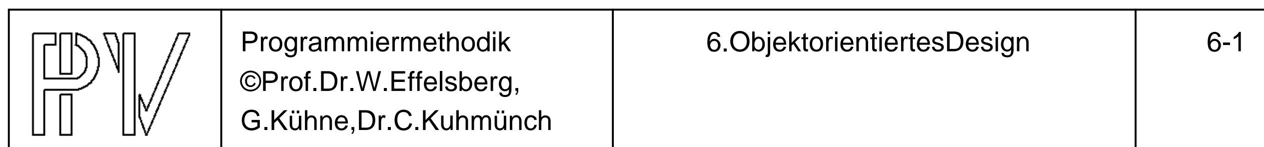
6.ObjektorientiertesDesign

6.1 Entwurfsmuster

6.2 ZusammenfassendesBeispiel

6.3 UmsetzungdesModel -View-Controller-MustersinJava



GrundlagedesobjektorientiertenDesigns

DasobjektorientierteDesignbestehtimWesentlichen ausdemAbbildendesinderOO -Analysegewonnenen Modellsauf **softwaretechnischeKlassen** .

TypischeKlassensind

- Klassen,dieHardware - undSoftware -Ressourcen repräsentieren
- GUI-Klassen,insbesonderefürdieErstellungvon Fenster-Oberflächen
- Middleware-Klassen
- AbstrakteKlassen
- Interface-Klassen
- Behälterklassen(Container, Collections)
- undvielemehr.

BeispielfüreineDatenbankanwendung

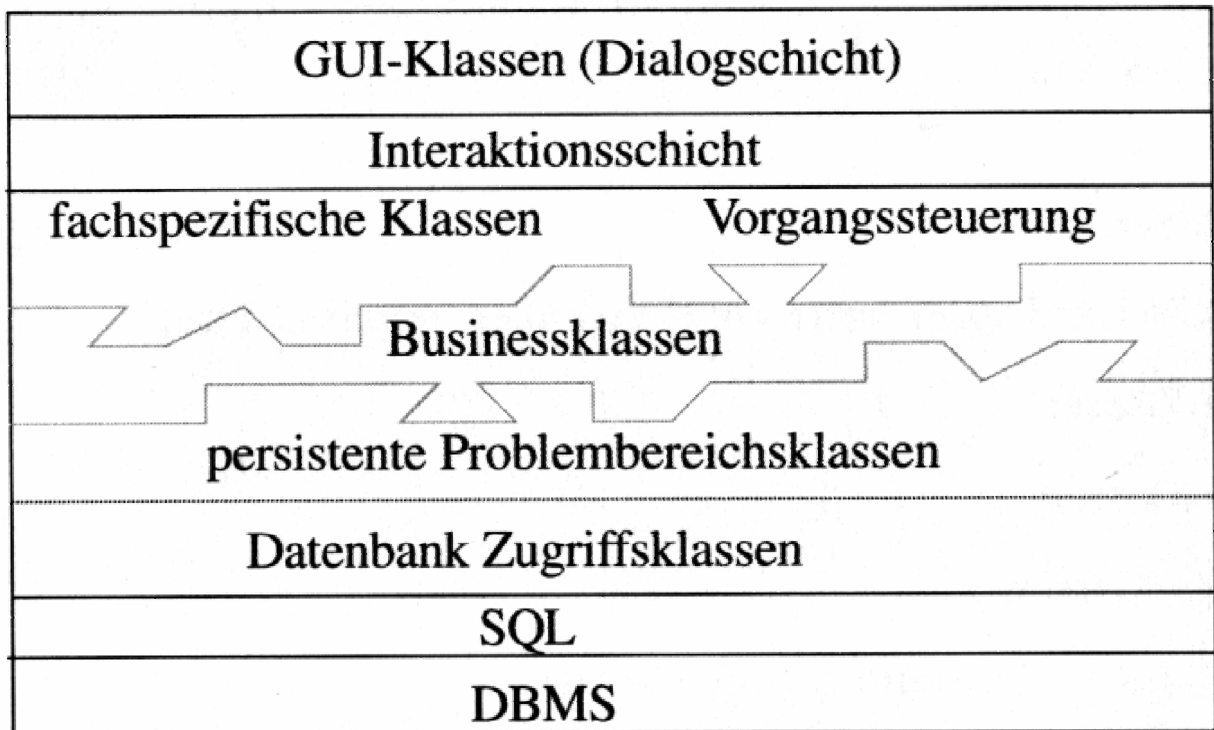


Abbildung 6.1 Architekturmodell

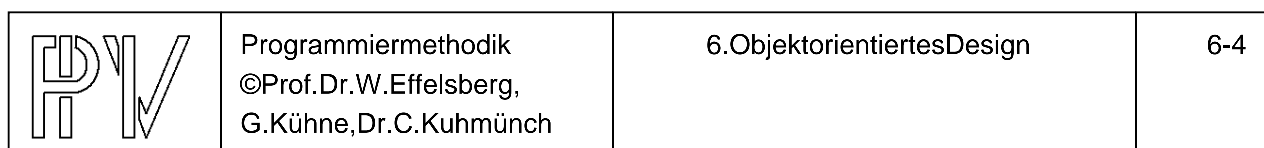
VieledieserKlassenwerdenKlassenbibliothekenoder Frameworksentnommenundbrauchennichtselbst implementiertzuwerden.Dasisteinwesentlicher UnterschiedzumkonventionellenSoftware - Engineering.

6.1Entwurfsmuster

NeuereArbeitenzumobjektorientiertenDesign schlagendieEinführungvon **Entwurfsmustern** (design patterns)vor.DieIdeeistdabei,dassmaninder Softwareentwicklungimmerwiederähnlichgeartete Problemelöst.

Entwurfsmustersindalso **verallgemeinerte Problemlösungskonzepte.** Essindkeine Klassenbibliotheken.

ImZusammenhangmitEntwurfsmusternistdie TrennungzwischenSchnittstelle(Interface)und Implementierungsinnvoll.EinesolcheTrennungkann inJavadurchabstrakteKlassenunddurchInterface - Klassenerfolgen.



Interface-Klassen

EineInterface -KlasseinJavaenthältnurdie DeklarationderMethoden,nichtihreImplementierung.

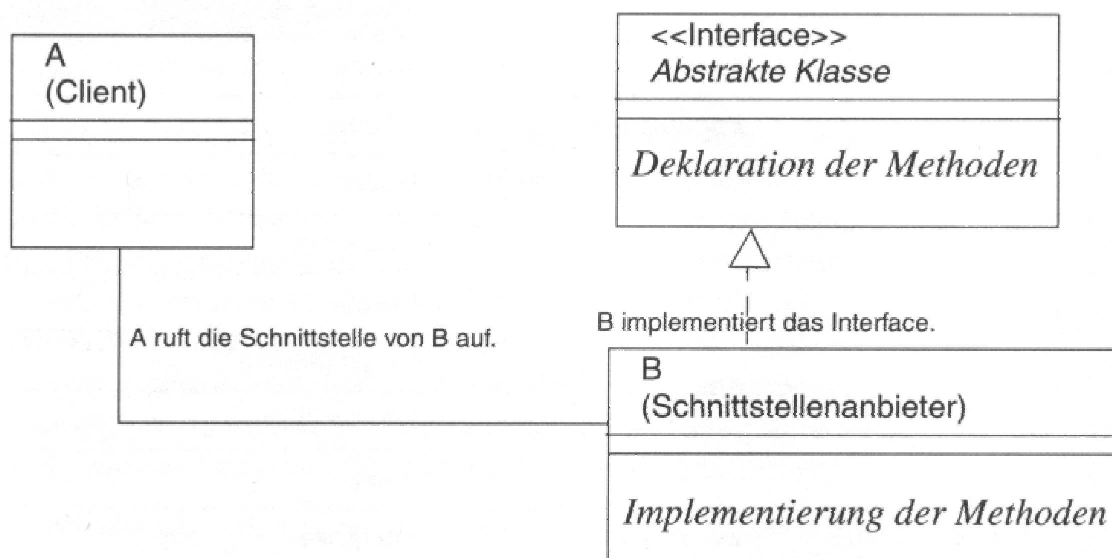


Abbildung 6.2 Trennung des Interface von der Implementierung

Model – View – Controller(1)

Model-View-Controlleristeinwichtigesundinder PraxisoftnützlichesEntwurfsmuster.

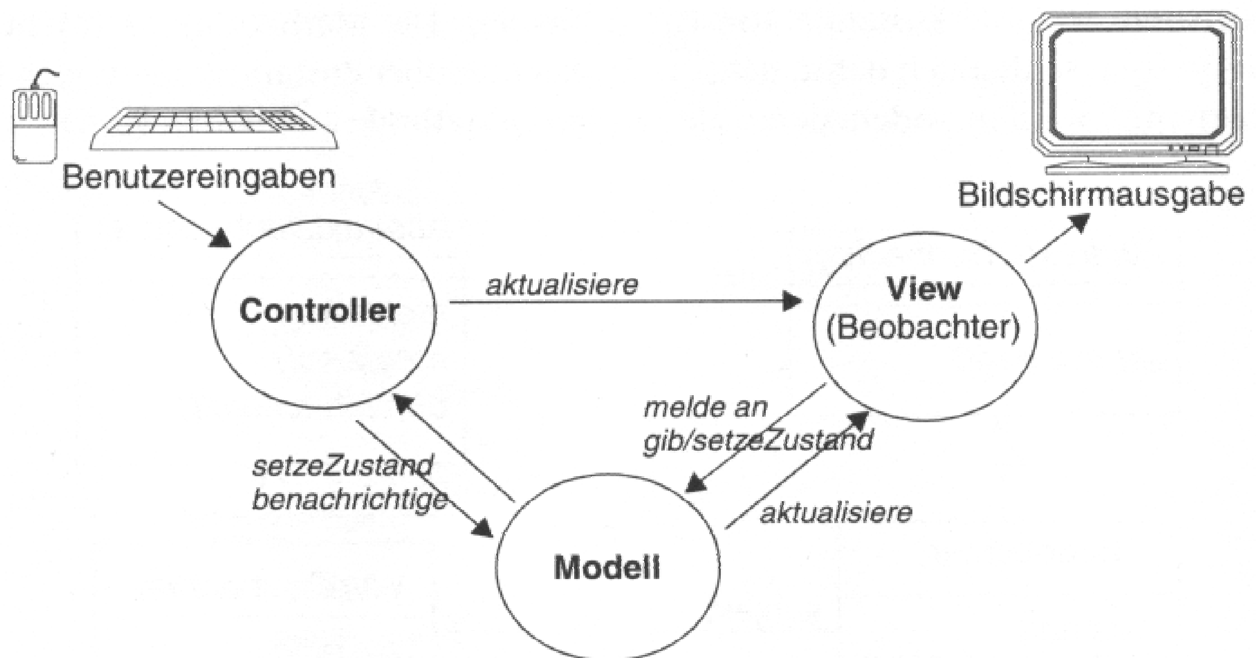


Abbildung 6.3 Modell View Controller

Model – View – Controller(2)

Model

ImModelistdieAnwendungslogikenthalten.Es
beschreibtdasSystemmodellundseingesamtes
immanentesVerhalten.

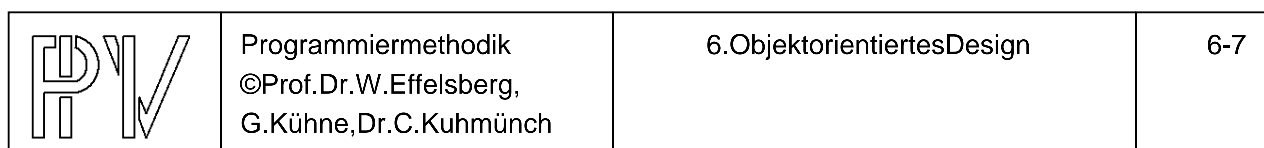
View

InderViewistdieRepräsentationdesZustandesdes
ModellsundseinerAusgabengegenüberdem
Benutzerzusammengefasst.

Controller

DerControllerimplementiertdieInteraktionmitdem
Benutzer,nimmtseineEingabenentgegen.

DiesaubereTrennungdieserdreiKomponentenhat
denVorteil,dassmaneineEinzelneleichtneu
implementierenkann,ohnedassdieanderenbetroffen
sind.



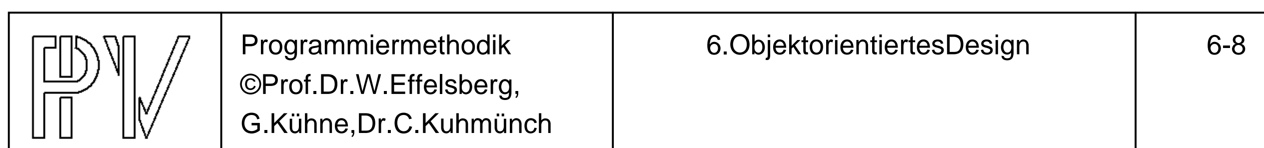
Das Beobachtermuster (1)

DasBeobachtermusterstellteineKopplungzwischen Beobachtern(View -Objekten)her,dieein observierbaresObjektbeobachten.

Das **beobachtbareObjekt** stelltfolgendeMethoden zurVerfügung:

- Anmelden
- Abmelden
- Zustandabfragen
- Zustandsetzen

Die **Beobachter** habeneineAktualisierungsmethode, dievombeobachtbarenObjektaufgerufenwird,wenn sichimbeobachtetenObjektetwasändert.



Das Beobachtermuster (2)

DieStrukturdesBeobachter -Musters

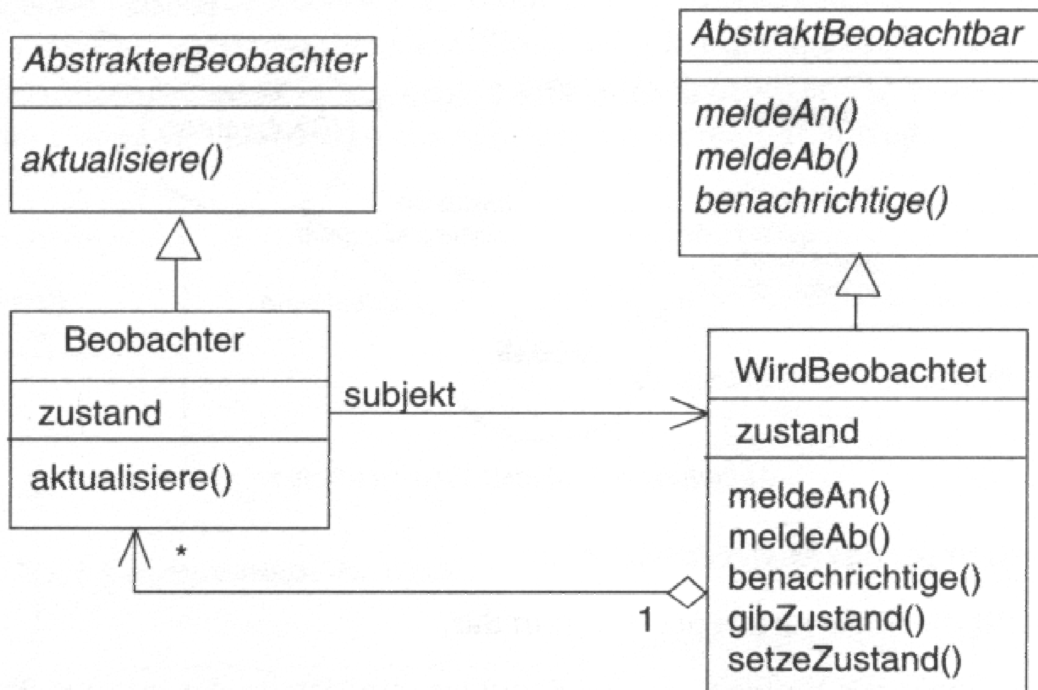


Abbildung 6.4 Beobachtermuster

InteraktionsdiagrammdesBeobachtermusters

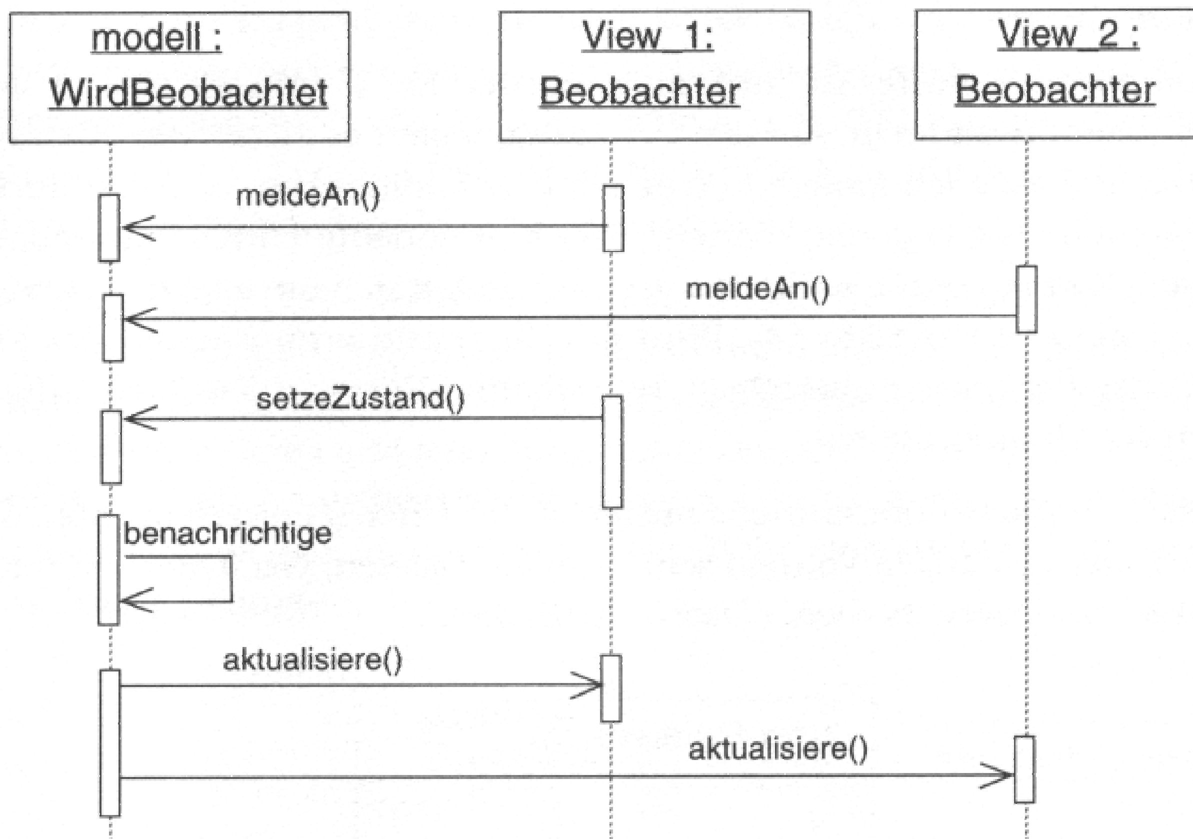


Abbildung 6.5 Interaktionsdiagramm für Beobachtermuster

DasKompositum

DasKompositumisteinEntwurfsmuster,dasRekursion zulässt.EsbeschreibtKomponenten,dieandere Komponentenenthaltenkönnen.

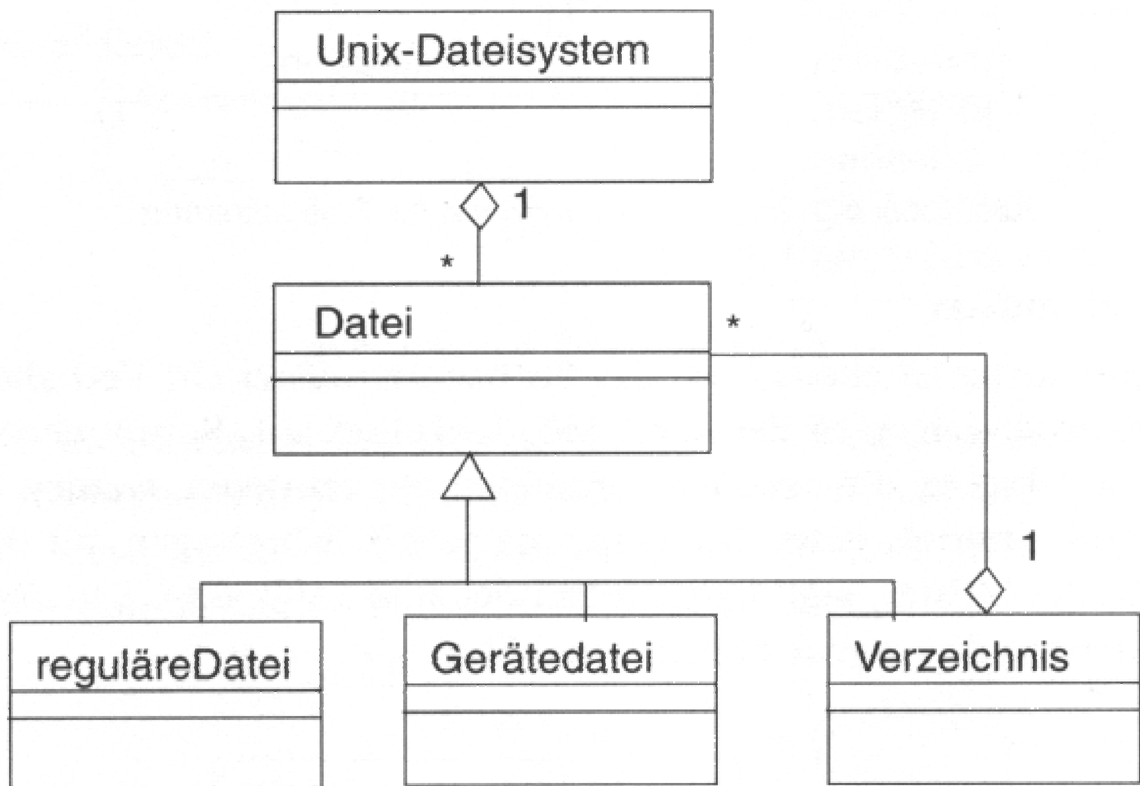


Abbildung 6.7 Beispiel für Kompositum

6.2ZusammenfassendesBeispiel

ReservierungvonRäumenfürLehrveranstaltungen

Use-CasesfürdasRaumreservierungsbeispiel

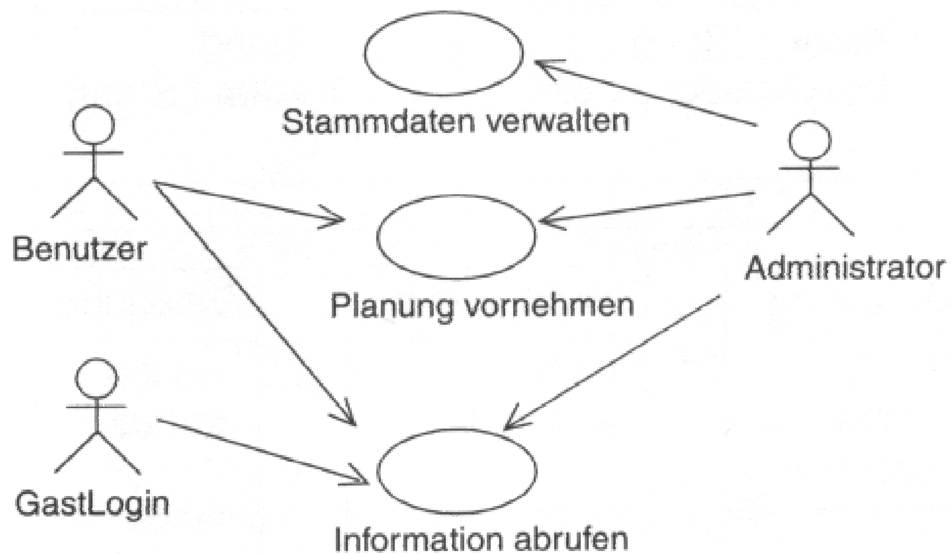


Abbildung 6.13 Use-Case-Szenario

Fachklassenfürdas Raumreservierungsbeispiel

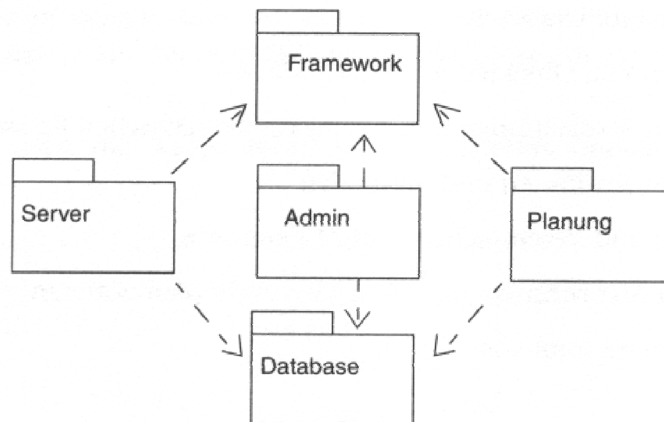


Abbildung 6.14 Pakete

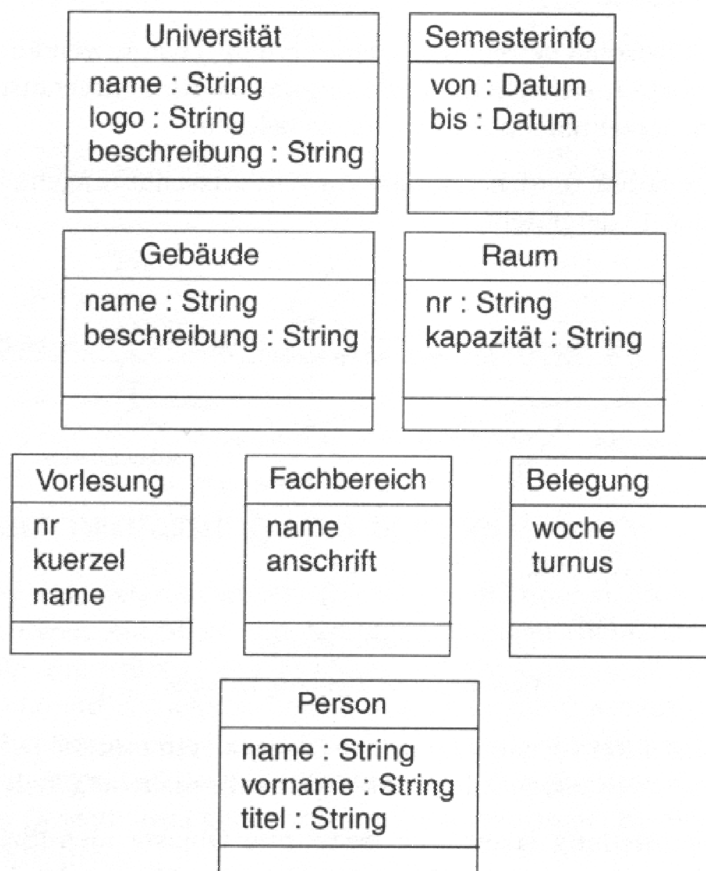


Abbildung 6.15 Fachklassen der Problemdomäne

BeziehungenzwischendenKlassendes Raumreservierungsbeispiels

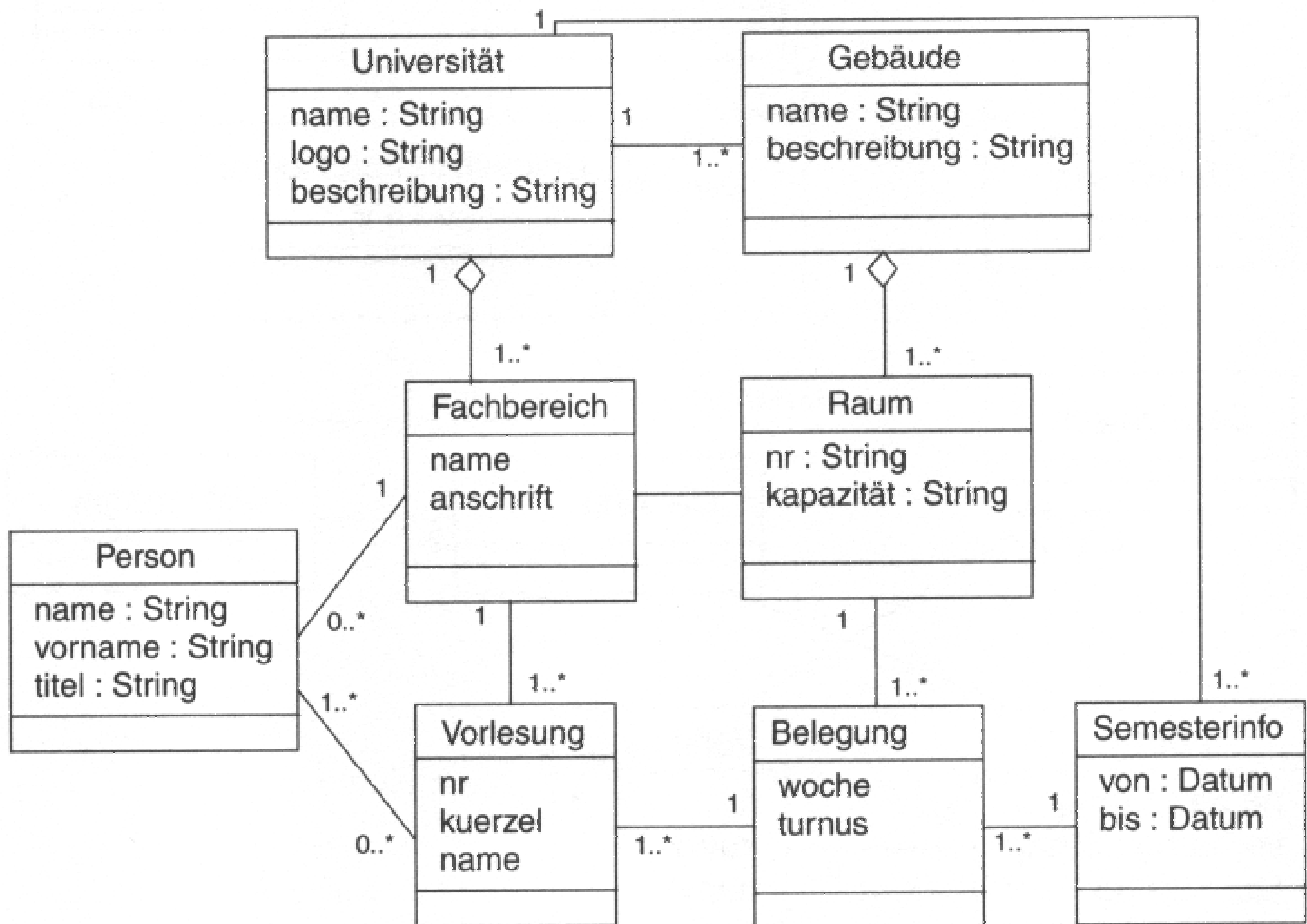


Abbildung 6.16 Beziehungen zwischen Klassen

RMI-KlassenfürdieVerteilung

BasisklassenfürdieVerteilungüberRMIhinzufügen

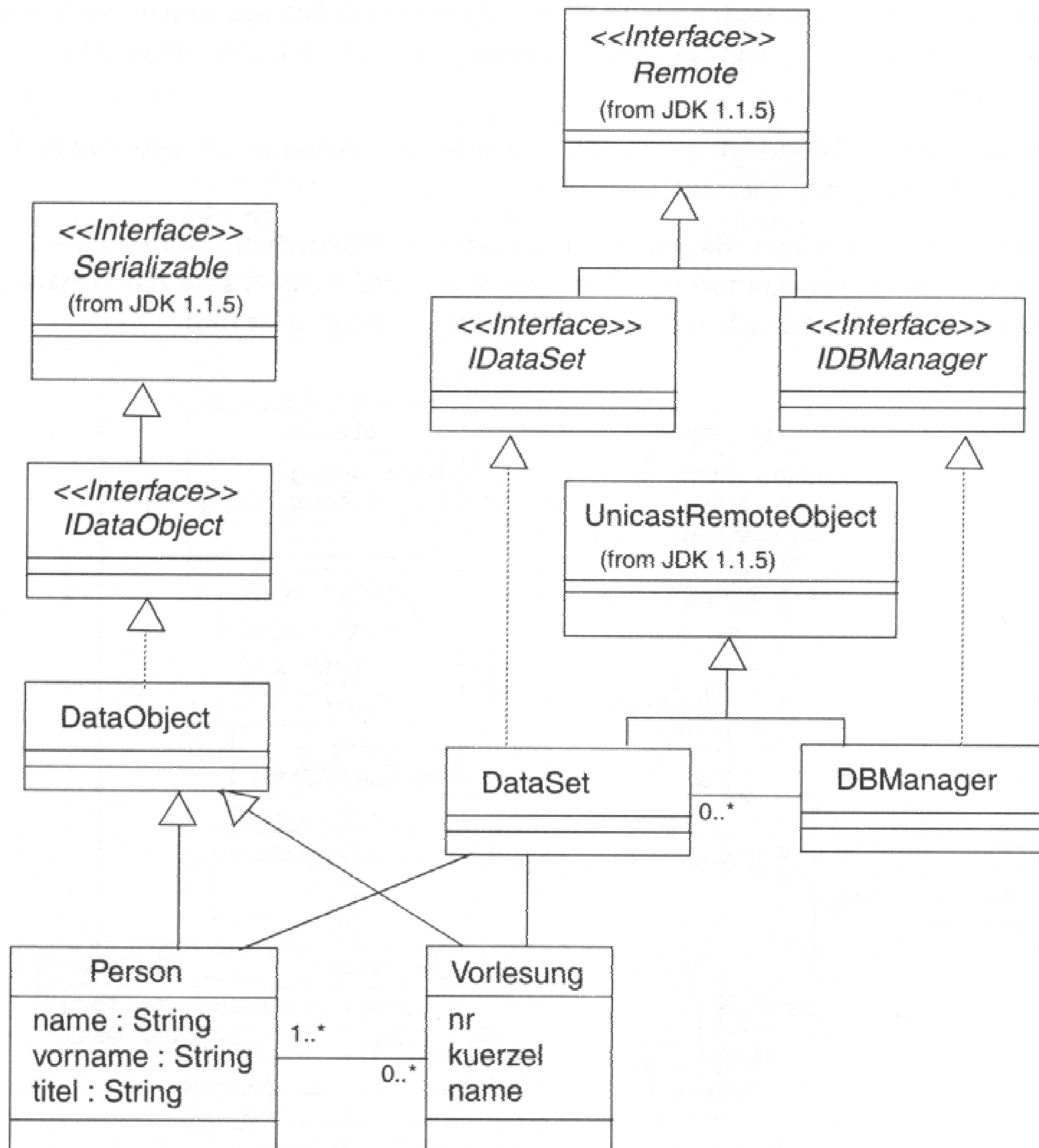
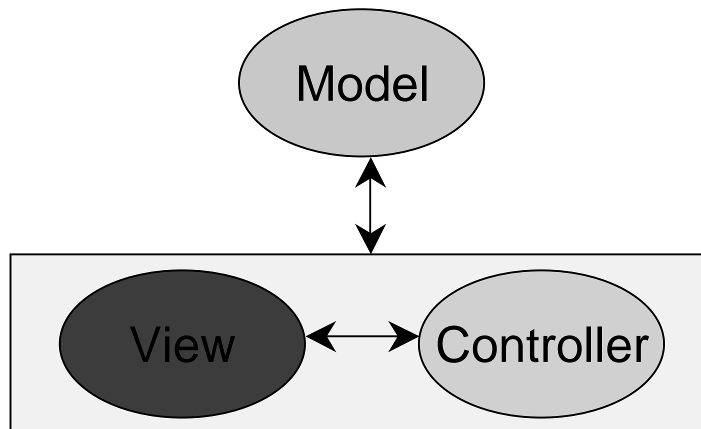


Abbildung 6.17 Basisklassen für Verteilung über RMI hinzufügen

6.3UmsetzungdesModel -View-Controller-MustersinJava

Model - Delegate-Architektur1/2

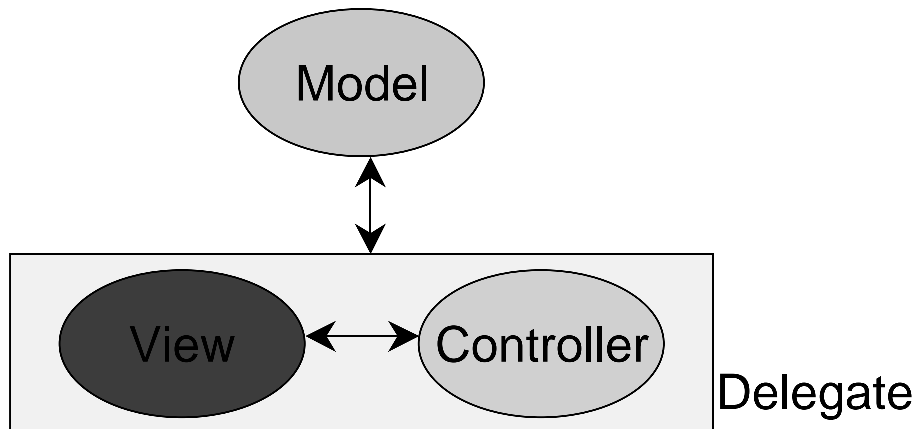


SwingimplementiertdieModel -Delegate-Architektur.

JedesDialogelementvonSwing(component)gliedertsich indasModell(model)unddieBenutzerschnittstelle(UI delegate).InnerhalbderBenutzerschnittstellekannman dieAnteilederBenutzeroberfläche(view)undder Anwen-dungssteuerung (controller)wiederfinden.

BeidieserStrukturierungverläuftdieKommunikation zwischendenKomponentenetwasandersalsbeidem MVC-Modell:EserfolgteineSynchronisationdes Zustan-desdesModellsunddesZustandesderBenutzerschnitt -stelle,diewahlweisevonbeidenSeitenausgelöstwerden kann.

Model - Delegate Architektur2/2

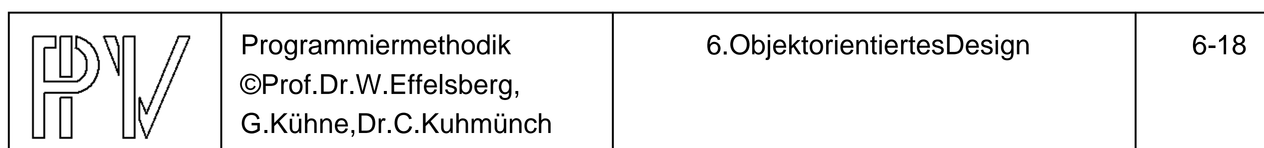


Dafürwerden `stateChanged-Events` verwendet. FernererfolgteindirekterAbgleichvonControllerund View durchandereEreignisse,dievonderjeweiligenArt desDialogelementsabhängigsind.DieseStrukturweist eineReihevonVorteilenauf:Siebindetdie Callback-FunktionenderAnwendungssteuerungskomponente engerandasjeweiligeDialogelement.Eshatsichnämlich herausgestellt,daßdasWiederverwendenvon Callback-FunktioneninunterschiedlichenKontextennurganz seltenmöglichist.(Beispieledafürsind:Beendeneiner Anwendung,SchließenallerFenster,aberdaswares meistauchschon.)

KommunikationzwischenModelund Delegate inJava

BisherkennenwirlediglichMethodenaufrufeals KommunikationsinstrumentzwischenzweiObjekten. Wolltenunein Delegate dasModelbspw.darüber informieren,daßeinbestimmterButtongedrückt wurde,müßteeswissen,welcheMethodewelchen Model-ObjekteszurBehandlungdieses *Ereignisses* notwendigist.UnterUmständenmüßtenmehrere Model-ObjektemitunterschiedlichenInterfaces informiertwerden.

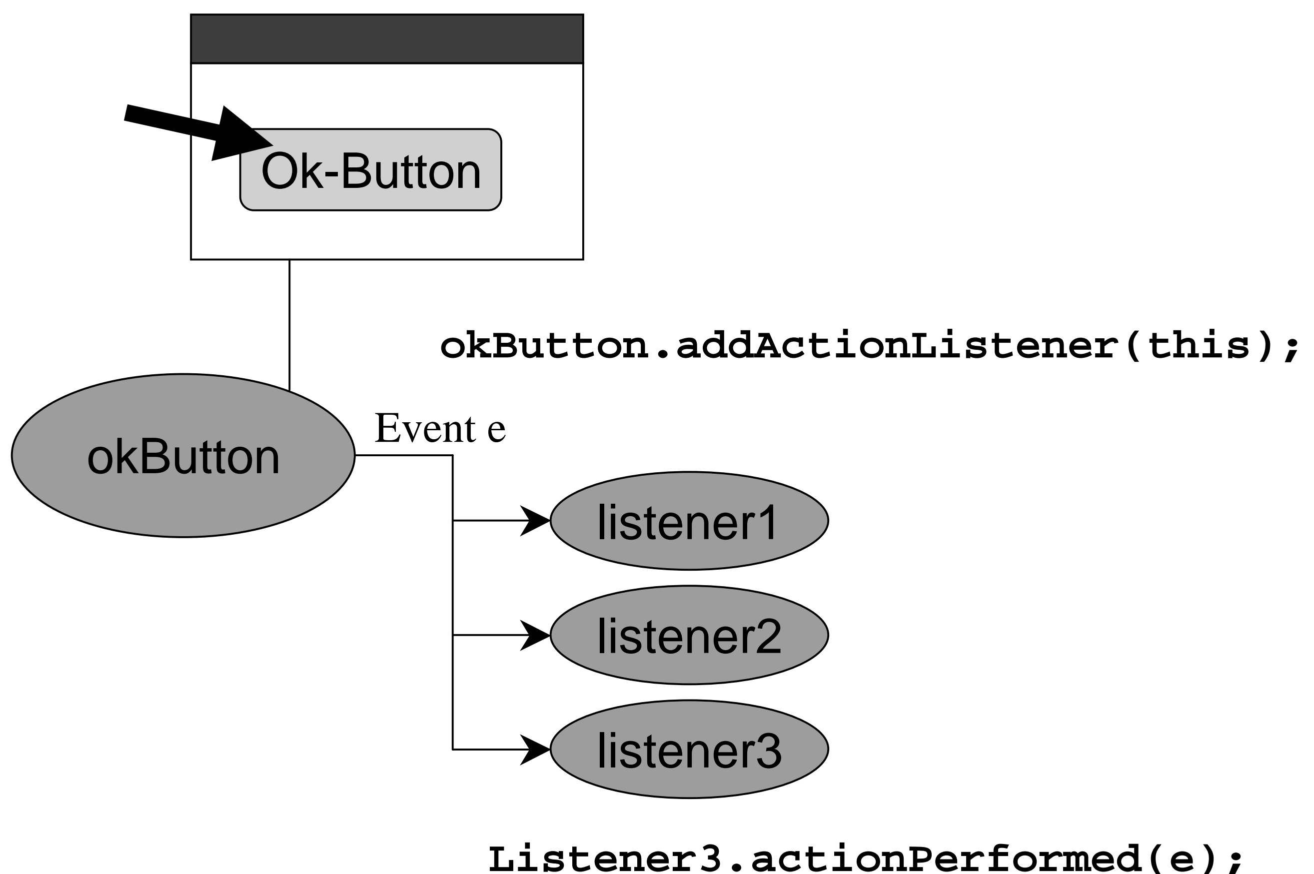
ZurVereinfachungdiesesProblemsimplementiert Javadas *Observer - Observable Muster*.



Observer - Observable Muster

DiesesMusterrealisierteine1:nVerknüpfungzwischen einemObjekt(Observable)undweiterendavon abhängigenObjekten(Observers).Eine ZustandsänderungdesObjekteshatdabeiautomatisch eineBenachrichtigungderanderenObjektezurFolge.

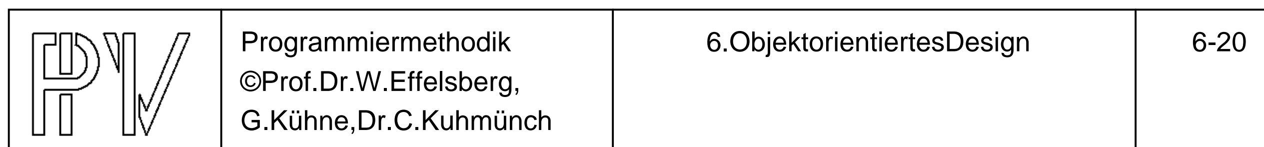
DieVerknüpfungzwischen   Observable undObservers kannunteranderemüberdasEreignis   -Konzept erfolgen.



Ereigniskonzept

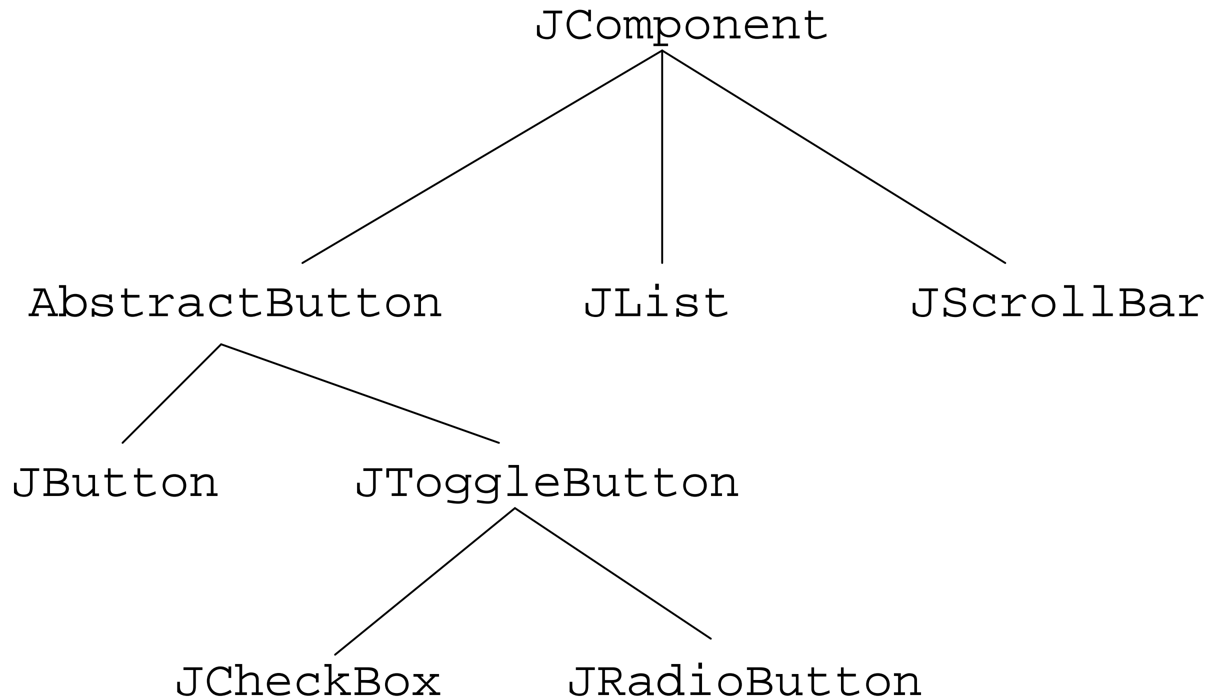
WesentlicheBegriffeinderProgrammierungvon BenutzeroberflächenmitSwingsinddieBegriffe: Component, Event und EventListener.

Unter Components werdenObjekteverstanden,die GUI-Elemente(Widgets)repräsentieren.EinBeispiel fürein Widget isteinButton.InteragiertderBenutzer miteinem Widget,erzeugtdieses Events. VerschiedeneBenutzerinteraktionenlösen verschiedene Events aus.Jede Component verwaltet Listenvon EventListeners fürverschiedene Event-Typen.Ein EventListener isteinObjekt,dassichfür ein Event einesbestimmtenTypsinteressiert.Um Events einerbestimmten Component zuempfangen, mußsichdiesesbeider Component anmelden.



Überblick Components

AuszugausderAbleitungshierarchieder Swing Components:



ZurRegistrierungvon EventListenern verfügen Components überMethoden

```
addXYZListener(XYZListener l);
```

Beispiel JButton

ObjektederKlasse JButton repräsentierenein Button Widget.DiesesverwalteteineListevon ActionEventListener Objekten,dieein ActionEvent erhalten,wennderBenutzerdenButtonklickt.

```
class JButton
  extends AbstractButton
{
 public void
  addActionListener(ActionListener l);
}
```

Beispiel ActionEventListener

DasDrückeneinesButtonslösteinEreignisvom Typ ActionEvent aus. ActionEvents könnenvon ActionListener Objektenverarbeitetwerden.

```
interface ActionListener
 extends EventListener
{
  public void
       actionPerformed(ActionEvent e);
};
```

BeispielAnwendung

NachfolgenderCodeauszugzeigt,wieeinObjekt ausdemProblembereichmiteinem Delegate verbundenwird.

```
class ButtonCtrl implements ActionListener
{
  public void
       actionPerformed(ActionEvent e)
    { System.out.println(“ok.“); }
}

public class MyClass
{
  public static ButtonCtrl bl;

  public static void main()
  {
    bl             = new ButtonCtrl();
    JButton    b   = new JButton(“Ok“);

    b.addActionListener(bl);
  }
}
```

Beispiel:„ PlayerList“

ErstellungeinesDialogfensterszur:

1. AnzeigederEinträgeundder Grösse einer Spielerlisteund
2. EingabevonweiterenSpielernamen

KomponentendesBeispiels

„Modell“

DasDatenmodellumfassteineListeinderdie Spielernamengespeichertwerden.

„View 1“

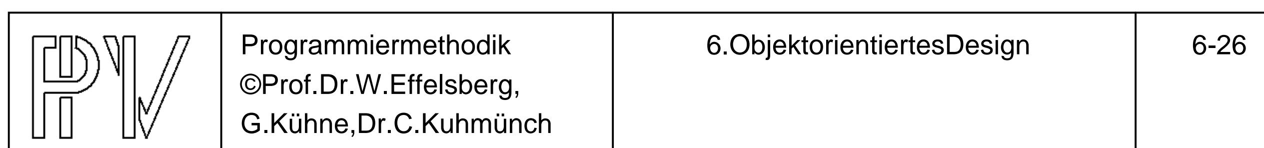
InnerhalbdererstenAnsichtwirddieListeder eingetragenenSpielernamenangezeigt.

„View 2“

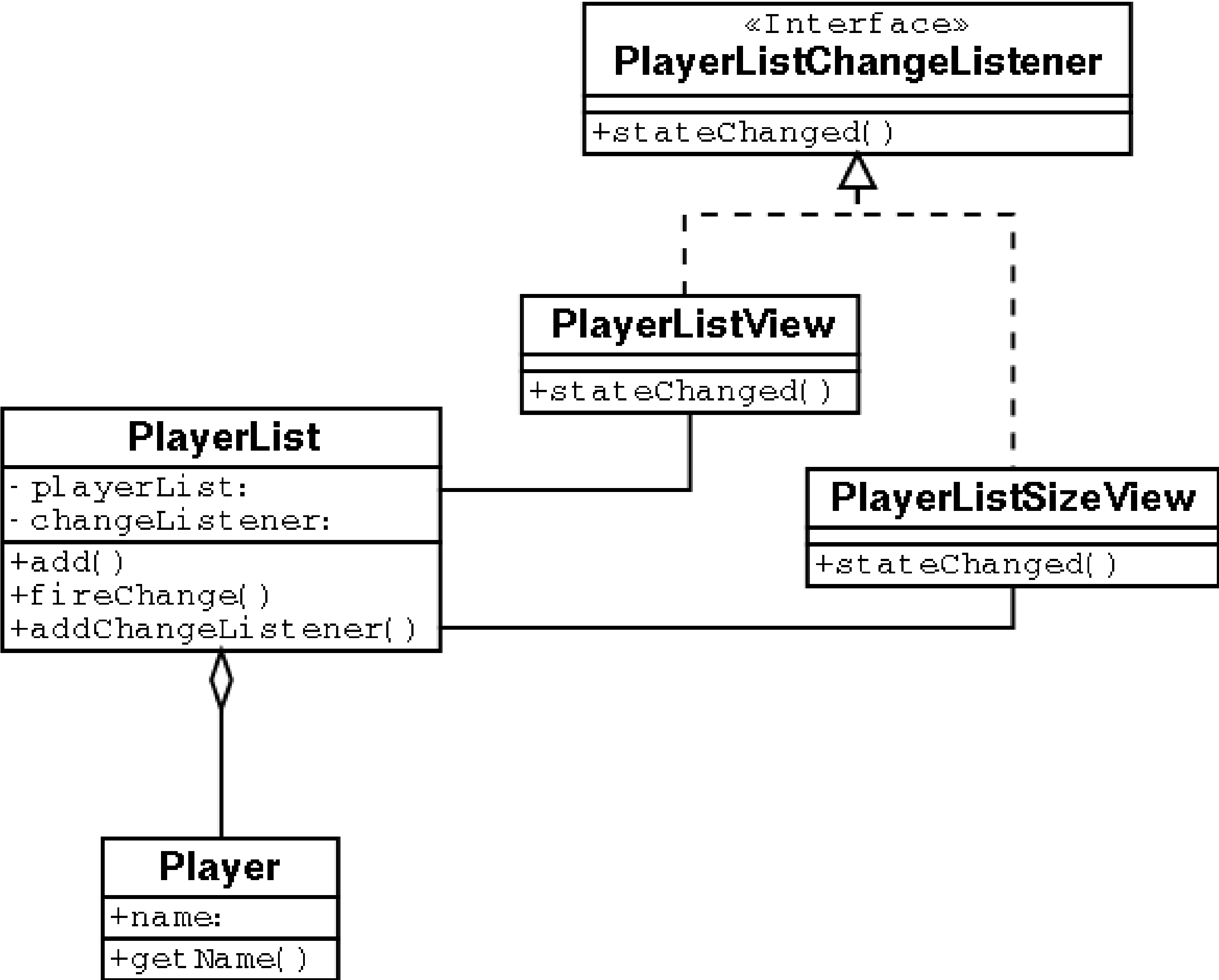
DiezweiteAnsichtumfasstdieAnzeigederaktuellen Grösse derSpielerliste.

„Controller“

DieBenutzereingabenwerdenüberdasTextfeld angenommenundentsprechendausgewertet.

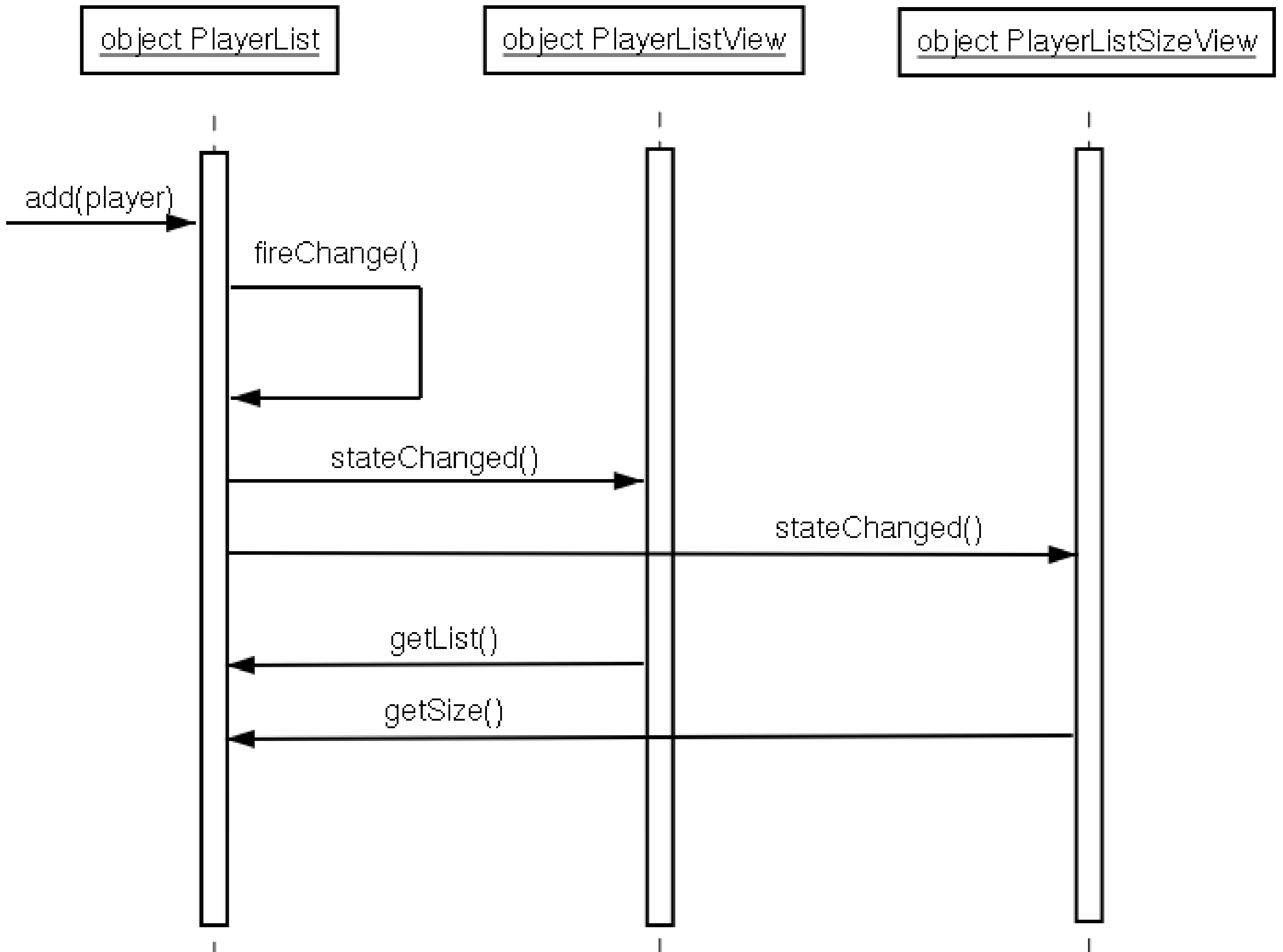


Klassendiagramm



Sequenzdiagramm

EreignisfolgebeimEinfügeneinesneuen Spielernamens:



Implementierungder Views (1)

Allgemeiner„ Listener“fürÄnderungender Spielerliste:

```
publicinterfacePlayerListChangeListener
{
   // thismethodiscalledwhentheplayer       list changes
   publicvoidstateChanged  ();
}
```

Implementierungder Views (2)

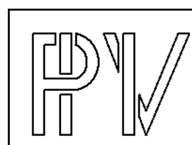
AnsichtzurAnzeigeder Listengrösse:

```
publicclassPlayerListSizeViewextendsJPanel
            implementsPlayerListChangeListener
{
   // theobservable
   PlayerListplayerList ;
   // thelabelthatshowsthesize
   JLabellabel  = newJLabel ();

   publicPlayerListSizeView (PlayerListplayerList )
   {
      this.playerList = playerList;

      // setthe  Swing component
      add(label);

      //update  thelabel
      stateChanged();
   }

   publicvoidstateChanged  ()
   {
      label.setText("Player list size :"+   playerList.getSize());
   }
}
```

Implementierungder Views (3)

AnsichtzurAnzeigederSpielernamen:

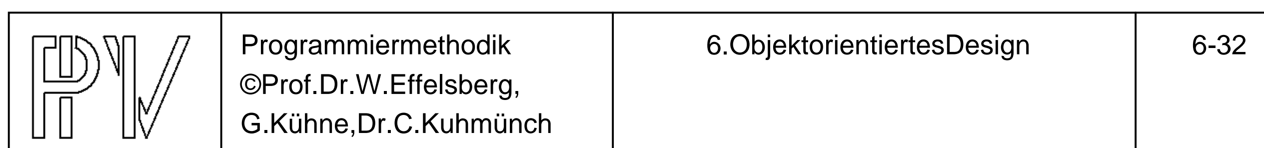
```
publicclassPlayerListViewextendsJPanel
            implementsPlayerListChangeListener
{
  JList list=  newJList (newDefaultListModel ());

  // player list thatisobserved
  PlayerListplayerList ;

  // construction
  publicPlayerListView (PlayerListplayerList )
  {
    // settheobservable
    this.playerList = playerList;

    setLayout(newBoxLayout (this, BoxLayout.Y_AXIS));

    // setthe  Swing components
    add(newJLabel ("Player list"));
    JScrollPanescrollPane  = newJScrollPane (list);
    add(scrollPane);

    //update  theJList
    stateChanged();
  }
```



Implementierungder Views (4)

AnsichtzurAnzeigederSpielernamen (Fortsetzung):

```
publicvoidstateChanged  ()
{
      // getthecurrent   list model
      DefaultListModeldlm  =( DefaultListModel)list. getModel();
      //and  clear all entries
      dlm.clear();

      // get all namesfromtheobserverplayer     list
      ListIteratorit  = playerList.getList().listIterator();
      while (it.hasNext()){
         Playercurrent  =( Player)it. next();
         // addthem  to the list model
         dlm.addElement(current.getName());
      }
   }
}//  classPlayerListView
```



ImplementierungdesModells

VerwaltungderSpielernamen:

```
publicclassPlayerList
{
   private ArrayListplayerList  = newArrayList ();
   private ArrayListchangeListener  = newArrayList ();

    publicvoidadd  (Player p){
        playerList.add(p);
        // notifylisteners
        fireChange();
    }

    publicArrayListgetList  (){  returnplayerList ;}
    publicintgetSize  (){          returnplayerList .size();}

    publicvoidfireChange  (){
        // notify all listenersthattheplayer    listhas  changed
        ListIteratorit  = changeListener.listIterator();
        while (it.hasNext()){
            PlayerListChangeListenercurrent  =
                        (PlayerListChangeListener) it.next();
            current.stateChanged();
        }
    }

    publicvoidaddChangeListener  (PlayerListChangeListener c)
    {  changeListener.add(c);}
}
```

IntegrationderKomponenten(1)

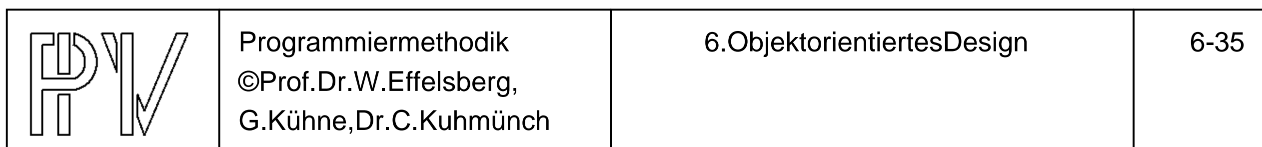
ErzeugungdesHauptfensters:

```
publicclass  Main
{
   // thedatamodel
   staticPlayerListplayerList   = newPlayerList ();

   // theinputfieldforaddingnewplayers
   staticJTextFieldtextField   = newJTextField ();

   publicstaticvoidmain   (String[] args)
   {
      // createtheapplicationwindow
      JFramemainFrame  = newJFrame ();
      Containerc=   mainFrame.getContentPane();
      c.setLayout(newFlowLayout ());
      ....
```

IntegrationderKomponenten(2)

Erzeugungder Views:

```
/* ---- createviews  ---- */

 // createthefirstview    ...
 PlayerListViewplv  = newPlayerListView (playerList);
 //...and   addit  to thelistener  listof  themodel
 playerList.addChangeListener(plv);

 // createthe  second view ...
 PlayerListSizeViewplsv  = newPlayerListSizeView (playerList);
 //...and   addit  to thelistener  listof  themodel
 playerList.addChangeListener(plsv);

 // integratetheviewsintotheapplicationframe
 JPanelviewPanel  = newJPanel ();
 viewPanel.setBorder(BorderFactory.createEtchedBorder());
 viewPanel.setLayout(newBoxLayout (viewPanel,
                              BoxLayout.Y_AXIS));
 viewPanel.add(newJLabel ("Views"));
 viewPanel.add(Box.createVerticalStrut(15));
 viewPanel.add(plv);
 viewPanel.add(plsv);
 c.add(viewPanel);
```



IntegrationderKomponenten(3)

ErzeugungdesControllers:

```
 /* ---- createcontroller  ---- */

// createthecontroller   (textField + ActionListener)
textField.addActionListener(newActionListener (){
      publicvoidactionPerformed  (ActionEvent e){
         playerList.add(newPlayer (textField.getText()));
         textField.setText("");
      }
   });

// integratethecontrollerintotheapplicationframe
JPanelcontrolPanel  = newJPanel ();
controlPanel.setBorder(BorderFactory.createEtchedBorder());
controlPanel.setLayout(newBoxLayout (controlPanel,
                                BoxLayout.Y_AXIS));
controlPanel.add(newJLabel ("Control"));
controlPanel.add(Box.createVerticalStrut(15));
controlPanel.add(newJLabel ("Enterplayername  :"));
controlPanel.add(textField);
c.add(controlPanel);

mainFrame.pack();
mainFrame.setVisible(true);
  }
}
```