OOD Case Study: Graphical-Based User Interface

21

The objective of this chapter is to apply the human–computer interaction (HCI) design concepts discussed in Chaps. 17 and 20 to the development of a graphical-based HCI.

21.1 OOD: Preconditions

The following should be true prior to starting this chapter.

- You understand the four criteria used to evaluate a HCI design:

1. Efficiency—How does the HCI affect the productivity of the user?
2. Learnability—How easy is it for a user to learn the HCI?
3. User satisfaction—How satisfied is the user with the HCI?
4. Utility—Does the HCI provide useful and timely information?

- You understand the HCI design goals: know the user, prevent user errors, optimize user abilities, and be consistent.
- You appreciate the value user participation brings to the HCI design process.
- Using Model–View–Controller for a text-based user interface means that the application controls the sequence of user interactions. This is done by having the controller component call view methods.
- Using Model–View–Controller for a graphical user interface means that the user controls the sequence of interactions. This is done by having the view component call controller methods.
- You understand the six characteristics of a good software design: simplicity, coupling, cohesion, information hiding, performance, and security.

D. P. Voorhees, *Guide to Efficient Software Design*, Texts in Computer Science,
https://doi.org/10.1007/978-3-030-28501-2_21

- You understand the notion of abstraction as a design process. Abstraction is the ability to generalize a concept by removing details that are not needed to properly convey the design perspective being emphasized.
- You have created and/or modified models that described an object-oriented or structured software design.
- You understand how to apply the Model–View–Controller architectural pattern to modify an existing object-oriented design into these three components.
- You understand how to apply the Model–View–Controller architectural pattern to create a high-level design that accurately reflects the domain requirements while satisfying the design constraints inherent in MVC.

21.2 OOD: ABA GUI Design

One GUI design is presented as an implementation of the ABA requirements, stated below for reference.

1. Allow for entry and (nonpersistent) storage of people's names.
2. Store for each person a single phone number and a single email address.
3. Use a simple text-based user interface to obtain the contact data.
4. Ensure that each name contains only uppercase letters, lowercase letters, and spaces. At least one letter must be entered for a name. There is no maximum limit on the size of the name entered.
5. Ensure that each phone number contains only the digits zero through nine. At least one digit must be entered for a phone number. There is no maximum limit on the size of the phone number entered.
6. Prevent a duplicate name from being stored in the address book.
7. Allow for display of contact data in the address book.

21.3 OOD: ABA GUI Version A

Two windows are used in this design, one lists all of the contact names while the other displays contact data for a single person. These two windows are shown in Figs. 21.1 and 21.2.

The statechart in Fig. 21.3 shows the actions the user may perform. First, the user will see contact names displayed in the list, representing contacts currently in the address book. Figure 21.1 lists four contact names. Second, the user can select an existing contact name and display the contact data for this person. Third, the user can click on the Create button to display the second window shown in Fig. 21.2. The user would enter appropriate data in the text fields and click OK to have the data validated. Valid contact data would result in the contact being added to the address book, assuming that the contact name does not already exist in the address book.

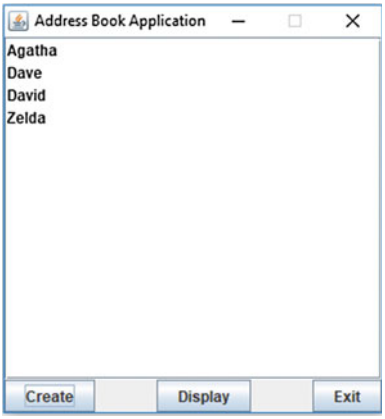

Fig. 21.1 ABA GUI Version A main window

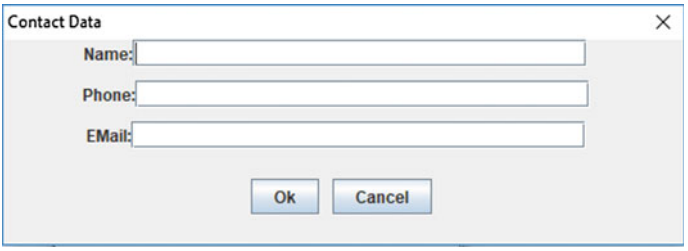

Fig. 21.2 ABA GUI Version A contact data

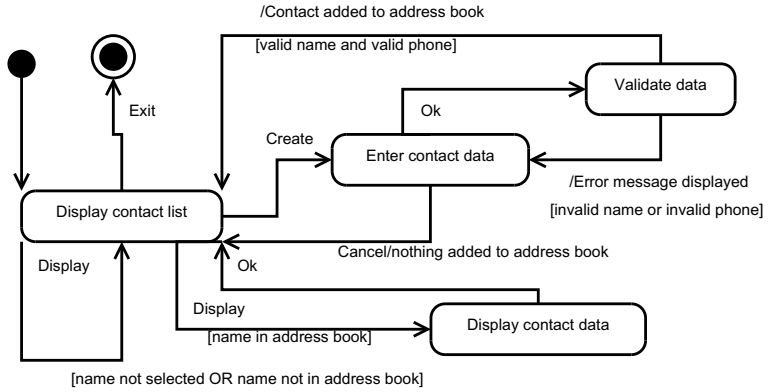

Fig. 21.3 ABA GUI Version A statechart

When the data is not valid, an error message is displayed and the user is allowed to correct the entered data. The user may then try to create the contact again. Finally, the user can exit the application.

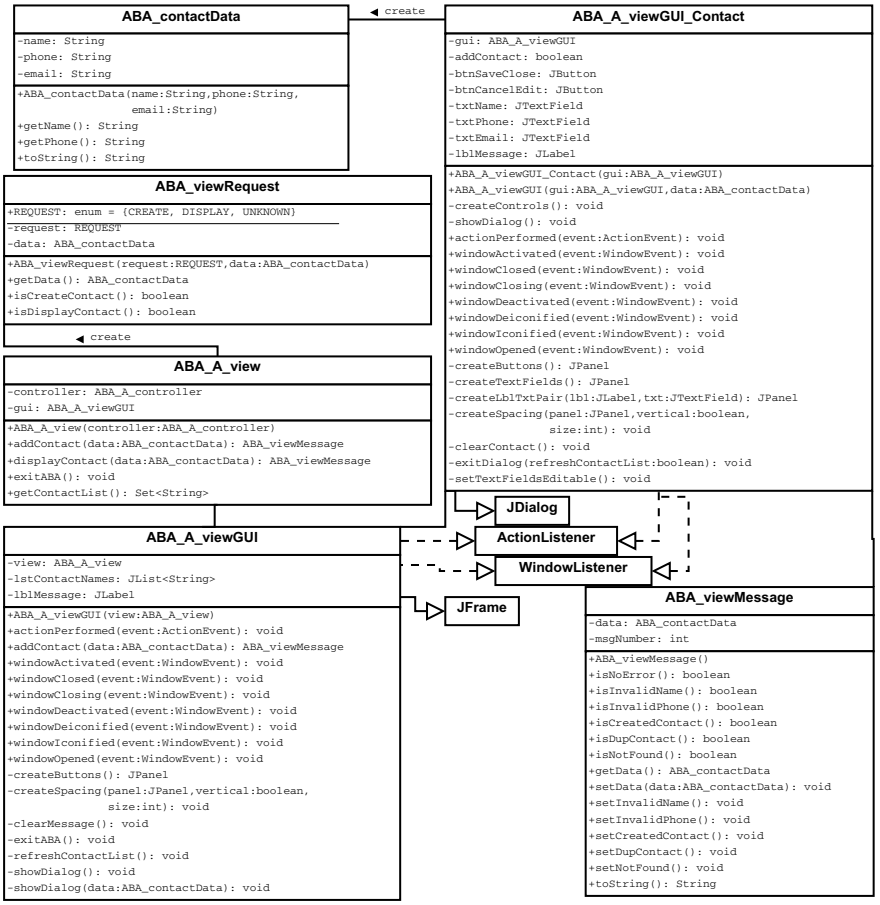

Fig. 21.4 ABA GUI Version A class diagram—view component

The classes that make up the view component are shown in Fig. 21.4. The viewGUI class contains the list of contact names along with buttons to create or display a specific contact. This class inherits from the JFrame Java API class and represents the main application window. The picture in Fig. 21.1 illustrates the appearance of this window. The viewGUI_Contact class contains text fields used to enter and display contact data for a single person. This class inherits from the JDialog Java API class indicating that it behaves like a pop-up window. More specifically, the viewGUI_Contact object is constructed as a *modal* dialog, meaning that while displayed, the user cannot interact with the application's main window. The picture in Fig. 21.2 illustrates the appearance of this dialog.

Figure 21.5 shows the UML communication diagram describing how the ABA Version A GUI application starts. The main method in ABA_A creates a ABA_A_controller object (see step 1). This constructor method first constructs a ABA_A_model

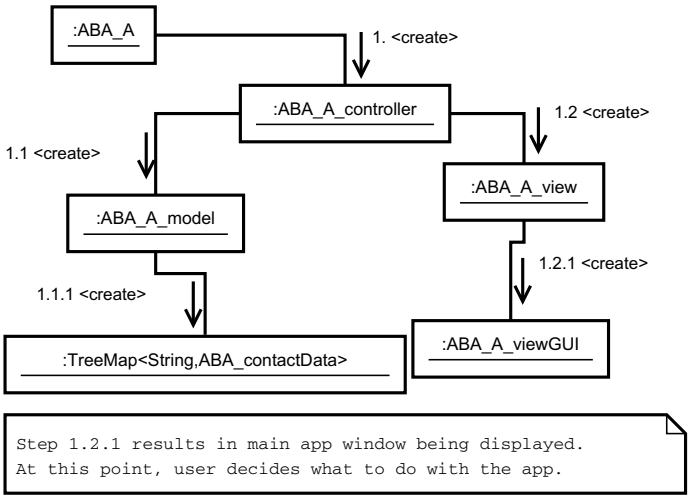

Fig. 21.5 ABA GUI Version A communication diagram—initializing ABA

object (step 1.1), then constructs a ABA_A_view object (step 1.2). The model constructor method creates a TreeMap object (step 1.1.1) used to store contact data. The view constructor method creates a ABA_A_viewGUI object (step 1.2.1) which is the main application window. At this point, the ABA is ready for use; the user determines what happens next! Note the absence of a go method in Fig. 21.5. As discussed in Chap. 20, this highlights one of the key differences between a text-based and graphical-based user interface; a GUI does not use a *driver* method while a TUI must have a method that drives the overall processing flow of the application.

One other detail about step 1.2.1 in the UML communication diagram shown in Fig. 21.5 is worth mentioning—the creation of the three buttons that appear on the main application window and how these are associated with a listener. Listing 21.1 shows portions of the ABA_A_viewGUI code illustrating how this is done in Java. The class implements two interfaces: ActionListener and WindowListener. A window listener deals with events sent to the JFrame, which includes the user clicking on the "X" button to close the application window. Listing 21.1 shows a WindowClosing method that will result in the ABA ending. The ABA_A_viewGUI code contains other WindowListener methods not shown in the Listing. An action listener is associated with individual user controls and requires the class to have an actionPerformed method (also partially shown in the Listing). The creation of the three push buttons shows a call to the addActionListener method passing *this*. Since *this* refers to the ABA_A_viewGUI object, these statements are indicating that any action on these buttons should be sent to the actionPerformed method defined in this class.

Listing 21.1 ABA MVC Design Version A - create JButton objects

```
public class ABA_A_viewGUI extends JFrame
        implements ActionListener, WindowListener
{
//not showing lots of code
   public void actionPerformed(ActionEvent event)
   {
      //not showing details of this method
   }
//not showing lots of code
   public void windowClosing(WindowEvent event) { exitABA(); }
//not showing lots of code
   JButton btnCreate = new JButton(CREATE_CONTACT);
   JButton btnDisplay = new JButton(DISPLAY_CONTACT);
   JButton btnExit = new JButton(EXIT_ABA);
   btnCreate.addActionListener(this);
   btnDisplay.addActionListener(this);
   btnExit.addActionListener(this);
//not showing lots of code
}
```

Figure 21.6 shows the UML communication diagram describing how the ABA Version A GUI creates a contact. This diagram assumes the user enters valid contact data to be added to the address book. The steps below describe the processing flow.

Step 1: The user clicks on the Create button on the main application window.
Step 1.1: The private showDialog method is called to display the pop-up dialog.
Step 1.1.1: An ABA_A_viewGUI_Contact object is created.
Step 1.1.1.1: The private createControls method is called to create the user controls in the dialog.
Step 1.1.1.2: The private showDialog method is called to show the dialog to the user.
Step 2: As noted, we assume the user enters valid contact data.
Step 3: We also assume the user clicks on the Save button within the dialog.
Step 3.1: The private createContact method is called.
Step 3.1.1: An ABA_contactData object is created containing the user-supplied data. The remaining steps refer to this object as data.
Step 3.1.2: The dialog tells the main application window (ABA_A_viewGUI) to add a contact using data.
Step 3.1.2.1: The main application window tells the view (ABA_A_view) to add a contact using data.
Step 3.1.2.1.1: The view creates an ABA_viewRequest object to communicate this user request to the controller. The remaining steps refer to this object as request.
Step 3.1.2.1.2: The view tells the controller ABA_A_controller to add a contact using request. The controller validates the user-supplied data, which is contained inside the request. In substep 1 the controller creates an ABA_viewMessage object to communicate the results of this request back to the view. The remaining steps

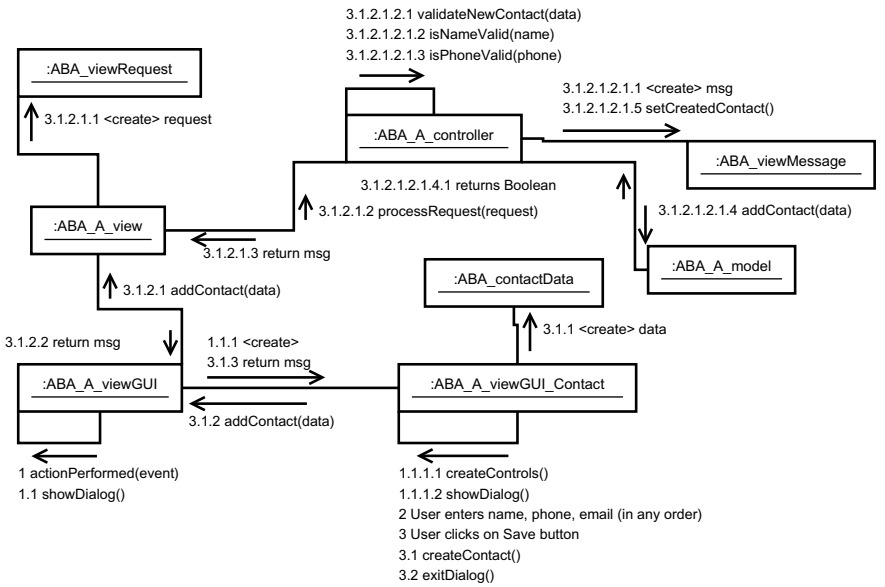

Fig. 21.6 ABA GUI Version A communication diagram—create a contact

refer to this object as msg. In substeps 2 and 2 the contact name and phone number are validated. Substep 4 adds the contact data to the address book; we assume the contact name is unique. In substep 5, the controller changes the state of the msg to indicate the contact was successfully added to the address book.

Step 3.1.2.1.3: The controller returns the msg to the view.

Step 3.1.2.2: The view returns the msg to the viewGUI.

Step 3.1.3: The viewGUI returns the msg to the viewGUI_Contact.

Step 3.2: The private method exitDialog is called to close the dialog and return the user to the main application window.

The processing flow to display existing contact data is similar. The viewGUI creates a new contactData object (aka data) containing the contact name associated with this display request. This data object is given to the view, which uses it to construct an ABA_viewRequest object (aka request). This request object is given to the controller, which then asks the model to retrieve the contact data for the user-supplied name. The controller creates an ABA_viewMessage object (aka msg) to communicate the results of this display request back to the view. This msg object is returned to the view, which returns it to the viewGUI (i.e, the main application window). The viewGUI then displays the viewGUI_Contact dialog showing the data retrieved by the model.

21.3.1 OOD: Evaluate Version A GUI Design

We'll now evaluate the Version A design using the HCI design criteria.

21.3.1.1 Efficiency

The display of existing contacts in the main application window allows the user to be more efficient in deciding whether a new contact needs to be created. Having the three buttons—Create, Display, and Exit—at the bottom of the window results in it taking a bit longer to activate one of these buttons. The design of the dialog box appears to be efficient, with two buttons when adding a contact but only one button when displaying existing contact data.

21.3.1.2 Learnability

The Version A GUI design is fairly easy to use. Two issues may cause a first-time user to be confused. First, it may take a moment before seeing the three buttons at the bottom of the main application window. Second, the large amount of space in the main application window with nothing being displayed may be initially confusing. On the positive side, the messages describing validation errors or success assist in the user's understanding of the application.

21.3.1.3 User Satisfaction

In order to assess this HCI design criteria, we would need to interview or survey users to obtain information on how satisfied they are with this GUI design. Since the ABA has not been deployed as a software application that others may use, this criteria cannot be assessed.

21.3.1.4 Utility

The user gets immediate feedback on each transaction they perform. Displaying a list of existing contacts is very useful to the user.

21.4 OOD Top-Down Design Perspective

We'll use the personal finances case study to reinforce design choices when creating a graphics-based user interface as part of a top-down design approach.

21.4.1 OOD Personal Finances: A Second Case Study

The requirements for personal finances, as stated in Chap. 12, are listed below.

- Allow a user to create as many accounts as they would like. Each account represents a single financial asset or liability that is provided as a financial service to an individual. Accounts are typically created to represent checking and savings accounts, credit card accounts, and loans for an automobile, school, or home. Each account has a name and a balance.
- Allow a user to enter as many transactions as they would like in each account. Each transaction represents a credit or debit within the account. Each transaction has a date, description, and amount (which can be a credit or debit). A transaction may optionally have a number (e.g., a check number for a checking account transaction) and zero or more labels.
- Allow a user to create labels, which are used to categorize a transaction. For example, labels may be created to indicate whether a transaction is associated with a charity, groceries, or home improvement.
- Allow a user to create as many reports as they would like. Each report includes one or more accounts, and zero or more labels.
- Allow a user to modify or delete any account, transaction, label, or report.

21.4.2 OOD Personal Finances: GUI Design

The user interface design for the personal finances case study is potentially quite large. It needs to support user actions to create, read, update, and delete accounts, transactions, and reports. Ideally, we would like a single window to provide quick access to each of these user actions.

Figure 21.7 shows a GUI design for the main application window. Accounts and reports would be listed along the left edge of the window. The majority of the main window will either show transactions for the selected account (shown in Fig. 21.7) or the selected report. Creating, updating, and deleting transactions would be done via choices on the Transactions menu. Similarly, accounts, reports, and labels may be created, updated, and deleted using these choices from the respective menu. The Labels menu also allows the user to display the entire list of labels.

The statechart and UML class diagram in Figs. 21.8 and 21.9 show the modifications to the view component as a result of using a graphical user interface for the personal finances application.

The statechart shows the user interactions associated with opening an account and adding, deleting, or updating a transaction. Once the transactions are displayed, the user selects an appropriate choice from the Transaction menu to initiate processing. When the user selects a different account from the list, the list of transactions associated with this account is displayed.

The UML class diagram includes two GUI classes (viewFinancesGUI and viewTransGUI) representing the main application window and the list of transactions displayed within this window. The viewUserRequest class is used to send a user request to the controller. A request can obtain data for display in the GUI (e.g., REQUEST_TYPE is VIEW), or send data to the controller for validation and ultimately for storage via the model. The viewAccount, viewLabel, viewReport, and

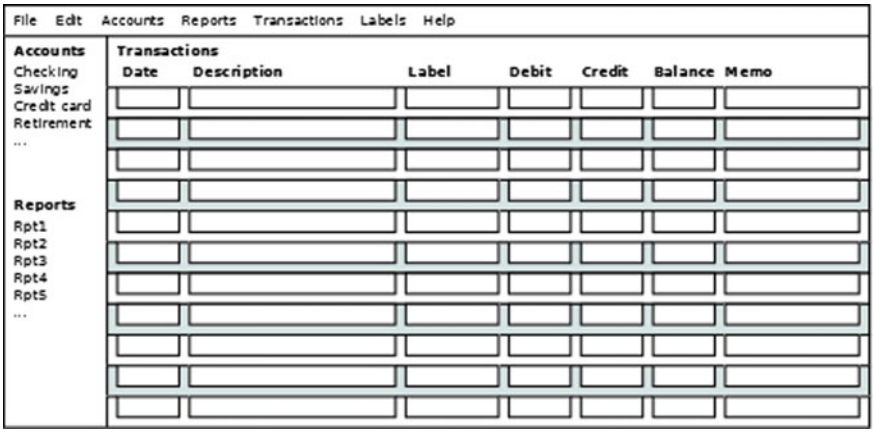

Fig. 21.7 Personal finances—GUI main window

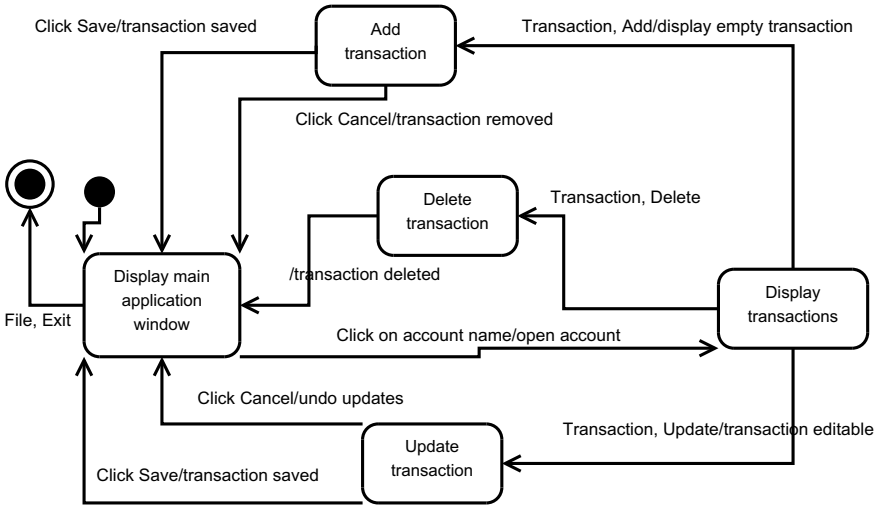

Fig. 21.8 State machine diagram—PF user interactions

viewTransaction classes are used to store a single instance of data either created by the user or obtained via the controller.

21.4.3 OOD Personal Finances: Evaluate GUI Design

With a single window supporting the application, efficiency, learnability, and utility may be assessed based on the design of this one window. As noted, the menu is used to create, update, or delete accounts, labels, reports, and transactions. Assuming

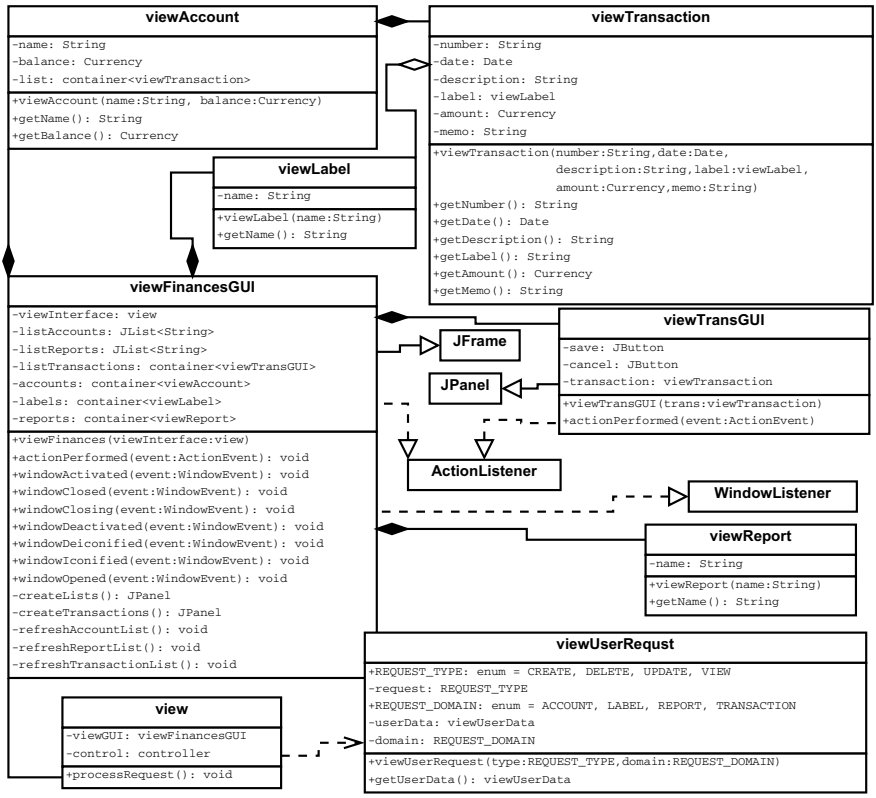

Fig. 21.9 Class diagram—PF view component

keyboard hot keys are associated with these menu choices, the user has the option of using a pointing device or the keyboard to activate an action. When an individual transaction is either created or updated, the GUI for that transaction will display save and cancel buttons. One of these buttons must be clicked to indicate the end of the create or update action.

From a learnability perspective, anyone familiar with a checking account transaction register, a small booklet used to record checking account transactions, will recognize the layout of transactions on the main application window. No attempt has been made to describe the GUI for creating, updating, or viewing a report. This is left as an exercise.

Given how similar the GUI design is to a paper-based checking account transaction register, the interface should provide the features necessary to support someone's personal finances.

21.5 OOD: Post-conditions

The following should have been learned when completing this chapter.

- A graphical user interface provides many alternatives for developing an HCI design. Creating a user interface design that is efficient, easy to learn, and useful is challenging because of the many choices one has for representing information in a graphical form.
- You understand how to apply the Model–View–Controller architectural pattern to modify an existing object-oriented design into these three components.
- You understand how to apply the Model–View–Controller architectural pattern to create a high-level design that accurately reflects the domain requirements while satisfying the design constraints inherent in MVC.
- You understand the six characteristics of a good software design: simplicity, coupling, cohesion, information hiding, performance, and security. You have evaluated a software design using these criteria.
- You understand that a UML class diagram, UML package diagram, IDEF0 function model, and data-flow diagram are design models used in object-oriented solutions to illustrate the structure of your software design.
- You understand that an IDEF0 function model, data-flow diagram, UML communication diagram, and UML statechart are design models used to illustrate the behavior of your software design.
- You have created and/or modified models that describe an object-oriented software design. This includes thinking about design from the bottom-up and from the top-down.

Exercises

Hands-on Exercises

1. Modify the GUI design and code for the ABA in any of the following ways.

   a. Allow a double-click on a contact name in the list to display the information for the selected contact.
   b. Move the buttons to the top of the main application window.
   c. Add hot keys to the buttons to create or display a contact.
   d. Change the JList to a JTable so the name, phone number, and email address can be displayed using three columns.

2. Modify the design for the personal finances application in any of the following ways.

   a. Show what a GUI would look like to create or update a report.
   b. Show what a GUI would look like to view a report.
   c. Update the class diagram based on your GUI designs for creating, updating, and viewing reports. That is, what is missing from the various classes to support these user actions?

3. Use an existing code solution that you've developed, develop alternative GUI-based HCI design models that show ways in which a user could interact with your application. Apply the HCI design criteria to your design models. How good or bad is your design?
4. Use your development of an application that you started in Chap. 3 for this exercise. Modify your HCI design to use some combination of graphical user controls, and then evaluate your design using the HCI design criteria.
5. Continue Hands-on Exercise 3 from Chap. 12 by developing an HCI design using some combination of graphical user controls. Be sure to develop design models to illustrate the structure and behavior of your design. The list below serves as a reminder of the application domain you may have chosen for this exercise. Refer to the Hands-on Exercises in Chap. 12 for details on each domain.

   - Airline reservation and seat assignment
   - Automated teller machine (ATM)
   - Bus transportation system
   - Course-class enrollment
   - Digital library
   - Inventory and distribution control
   - Online retail shopping cart
   - Personal calendar
   - Travel itinerary