18 OOD Case Study: Text-Based User Interface

The objective of this chapter is to apply the human–computer interaction (HCI) design concepts discussed in Chap. 17 to develop a better text-based HCI.

18.1 OOD: Preconditions

The following should be true prior to starting this chapter.

- You understand four HCI design criteria: efficiency, learnability, user satisfaction, and utility. You have evaluated user interface designs using these criteria.
- You understand that the process of creating an HCI design is critical for developing a user interface that satisfies the four criteria listed above. While developing an HCI design, the designer should know the user, prevent user errors, optimize user abilities, and be consistent.
- You understand Model–View–Controller as a software architecture that separates the user interface (view) from the application data (model). This separation is achieved by putting the domain logic in the controller and enforcing constraints on how these three components communicate with each other.
- You understand five program design criteria: separation of concerns, design for reuse, design only what is needed, performance, and security. You have evaluated program code using these criteria.
- 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.

D. P. Voorhees, *Guide to Efficient Software Design*, Texts in Computer Science,
https://doi.org/10.1007/978-3-030-28501-2_18

- 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.

18.2 OOD: Concepts and Context

Even though most modern software use a touch screen and/or graphics-based user interface (UI), there is still a need to use a more primitive text-based HCI. Examples of text-based user interfaces (TUI) include the *Terminal* window in Unix-like systems, the *Command Prompt* window in Windows-based systems, configuration of operating system components (e.g., BIOS settings), and many text-based games that were popular in the 1980s.

18.2.1 OOD: TUI Design Alternatives

There are a few common text-based UI approaches that may be combined to create a text-based software application. The alternative TUI design approaches include the following.

Guided Prompts: The user is prompted to enter a specific data value. This is a common design choice when the software needs to obtain data from the user. The ABA case study seen so far uses this approach.

Menus: A list of menu choices is displayed to the user followed by the user entering a letter or digit matching one of the menu choices. Once a valid menu choice has been entered, the associated processing is initiated. Many of the TUIs designed to configure an operating system component will utilize menus.

Commands: A command prompt is displayed to the user allowing the user to enter a command. Once a valid command has been entered, the associated processing is initiated. The Terminal window (in Unix-like systems) and the Command Prompt window (in Windows systems) are examples of this approach.

18.3 OOD: ABA TUI Designs

The current TUI design provides an inflexible interface to the user. The user must first create new entries in the address book. Only after exiting *data entry mode* does the user see the contents of the address book. This inflexible user interface design is shown in the state machine diagram from Chap. 15, shown in Fig. 18.1.

Two text-based UI designs are described below. Both provide the user with the flexibility to switch between creating new address book entries and displaying existing address book entries. The first design combines menus with guided prompts for

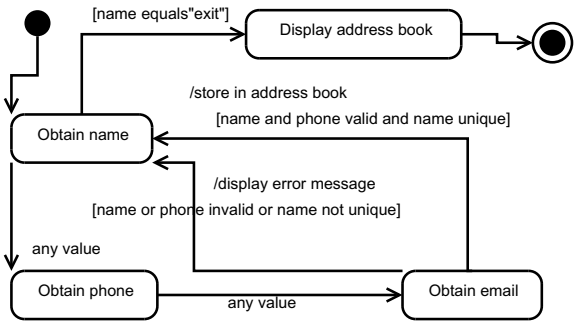

Fig. 18.1 State machine diagram—ABA Version B Chap. 15

data entry while the second design combines commands with guided prompts for data entry.

18.3.1 OOD: Menu and Guided Prompts

The sample user interactions shown in Listing 18.1 uses a menu to display the choices available to the user and guided prompts for entry of data values. This particular example shows the user creating a new address book entry, displaying the just created address book entry, and then exiting the application.

Listing 18.1 Sample ABA Menu & Data Entry Prompts

```
C   Create address book entry
D   Display address book entry
X   Exit
Choice: C

Enter contact name: Santa Claus
Enter phone number for Santa Claus: 3155559876
Enter email address for Santa Claus: santa@claus.org

C   Create address book entry
D   Display address book entry
X   Exit
Choice: d

Enter contact name: Santa Claus
The phone number for Santa Claus is 3155559876
The email address for Santa Claus is santa@claus.org

C   Create address book entry
D   Display address book entry
X   Exit
Choice: x
```

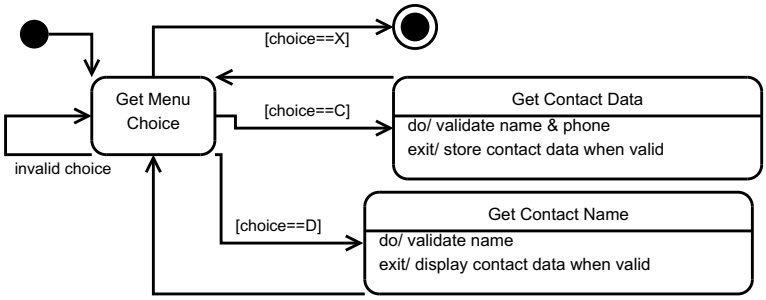

Fig. 18.2 State machine diagram—ABA menu and data entry prompts

The state machine diagram in Fig. 18.2 shows how the use of a menu provides more flexibility to the user in terms of switching between creating a new address book entry and displaying existing entries. The *Get Menu Choice* state will validate the menu choice entered by the user. When a valid menu choice is entered, the user is transitioned to a state that is appropriate for the valid menu choice. After the user has either done *Get Contact Data* or *Get Contact Name*, the user transitions back to the *Get Menu Choice* state to allow the user to decide what to do next.

18.3.2 OOD: Commands and Guided Prompts

The sample user interactions shown in Listing 18.2 uses a command prompt to obtain commands from the user and guided prompts for entry of data values. This particular example shows the user requesting help, creating a new address book entry, displaying the just created address book entry, and then exiting the application.

Listing 18.2 Sample ABA Commands and Data Entry Prompts

```
command> help
All commands are case insensitive.
Enter 'C' or 'create' to create an address book entry.
Enter 'D' or 'display' to display an address book entry.
Enter 'H' or 'help' to display this help text.
Enter 'X' or 'exit' to exit.

command> C
Enter contact name: Santa Claus
Enter phone number for Santa Claus: 3155559876
Enter email address for Santa Claus: santa@claus.org

command> display
Enter contact name: Santa Claus
The phone number for Santa Claus is 3155559876
The email address for Santa Claus is santa@claus.org

command> X
```

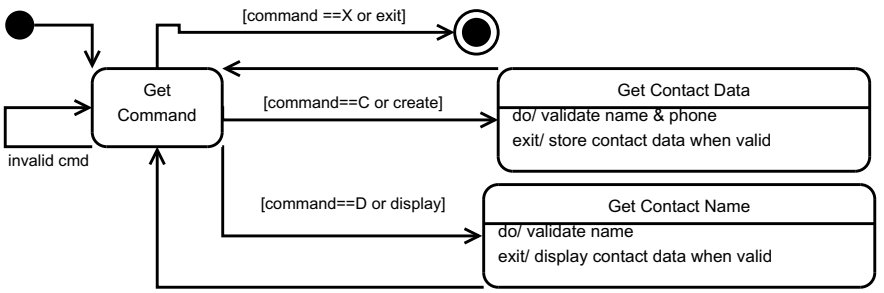

Fig. 18.3 State machine diagram—ABA command and data entry prompts

The state machine diagram shown in Fig. 18.3 shows how the use of a command prompt provides more flexibility to the user in terms of switching between creating a new address book entry and displaying existing entries. The *Get Command* state will validate the command entered by the user. When a valid command is entered, the user is transitioned to a state that is appropriate for the valid command. After the user has either done *Get Contact Data* or *Get Contact Name*, the user transitions back to the *Get Command* state to allow the user to decide what to do next. This design includes a command to display help information.

18.3.3 OOD: Evaluate ABA TUI Designs

To assess the three TUI designs—the Chap. 15 Version B design and the two alternative designs described above—we will focus on those aspects of the designs that differ from each other. The entry of contact data to create an address book, which is the same in all three designs, will be ignored in the descriptions below.

18.3.3.1 Efficiency

How does the HCI design affect the productivity of the user? Table 18.1 summarizes the differences in efficiency. The flexibility provided by the two designs presented in this chapter allows the user to more efficiently use the ABA. The entry of a menu choice or command does not adversely affect the assessment that the two designs introduced in this chapter are more efficient than the Chap. 15 design.

18.3.3.2 Learnability

How easy is it for a user to learn the HCI? Table 18.2 summarizes the differences in learnability. The Chap. 15 UI and the menu and guided prompts designs are easier to learn than the command and guided prompts design.

Table 18.1 Efficiency comparison

Chapter 15 UI	When a user wants to use the ABA to display existing contact information, they must enter "exit" as a contact name, as shown in Fig. 18.1
Menu and guided prompts	A user must enter a menu choice but can switch between creating and displaying contact data as often as they would like, as shown in Fig. 18.2
Commands and guided prompts	A user must enter a command but can switch between creating and displaying contact data as often as they would like, as shown in Fig. 18.3

Table 18.2 Learnability comparison

Chapter 15 UI	The sequential nature of this design, i.e., a user must first create address book data and explicitly "exit" before displaying address book data, makes this design easy to learn how to use
Menu and guided prompts	The menu choices self-document the options available to the user, making this design easy to learn how to use
Commands and guided prompts	A power user would know the commands and not need the help facility. Someone first learning or rarely using the ABA would likely need to display the help information a few times before remembering the commands

18.3.3.3 User Satisfaction

How satisfied is the user with the HCI? 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 each of the design alternatives. Since the ABA has not been deployed as a software application that others may use, this criteria cannot be assessed.

18.3.3.4 Utility

Does the HCI provide useful and timely information? Table 18.3 summarizes the differences in utility. The flexibility provided by the two designs presented in this chapter allows the user to use the ABA in a manner that is more useful to them by providing timely access to information.

18.3.4 OOD: Commands Design Revisited

The TUI design that uses commands is particularly useful when many of your users are considered power users. In this case, we can extend the definition of the commands to include the appropriate address book data. The sample user interactions shown

Table 18.3 Utility comparison

Chapter 15 UI	The sequential nature of this design, i.e., a user must first create address book data and explicitly "exit" before displaying address book data, results in the ABA being of little use
Menu and guided prompts	The flexibility afforded a user in switching between creating and displaying contact data as often as they would like makes this design useful
Commands and guided prompts	The flexibility afforded a user in switching between creating and displaying contact data as often as they would like makes this design useful

in Listing 18.3 use more complex commands, allowing a command to optionally include user-supplied data typically obtained via the guided prompts.

Listing 18.3 Sample ABA Commands Revisited

```
command> help
All commands are case insensitive.
Command parameters in [square brackets] are optional.
Can use single (') or double quotes (") to delimit a contact name.
To create an address book entry: c ['name'] [phone] [email]
                                 create ['name'] [phone] [email]
To display an address book entry: d ['name']
                                  display ['name']
To display this help text: h
                           help
To exit: x OR exit

command> c 'Santa Claus' 3155559876 santa@claus.org

command> c "Jane Claus"
Enter phone number for Jane Claus: 3155551209
Enter email address for Jane Claus: jane@claus.org

command> display "Santa Claus"
The phone number for Santa Claus is 3155559876
The email address for Santa Claus is santa@claus.org

command> x
```

18.3.4.1 Evaluating Commands Design Revisited

The evaluation of this updated version of the Commands and Guided Prompts design is described below.

Efficiency: The efficiency of this updated commands design is superior to any of the other three designs since a *create* or *display* command may include all of

the necessary user-supplied address book data values. This eliminates the guided prompts for obtaining a name, phone number, and email address.

Learnability: This updated commands design is harder to learn than the other three designs since the commands are more complex. As shown when submitting a *help* command, the help information is much more complex than the earlier version of the commands design. As already mentioned, when you have users that are comfortable entering more complex commands (i.e., you have power users), this updated commands design may be your best alternative for implementation.

Utility: This updated commands design is just as useful and provides as timely information as the two designs presented earlier in this chapter.

18.4 OOD Case Study: TUI Design Details

A requirement has been added to the ABA, shown in the listing below in *italics*.

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 upper case letters, lower case 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.*

A text-based user interface requires the software to guide the user through the steps needed to complete a processing step. For example, the *Get Contact Data* state in the state machine diagrams in Figs. 18.2 and 18.3 forces the user to enter a contact name, then a phone number, and then an email address. The TUI design does not allow the entry of these values in any other order. This sequence of data entry steps is shown in Fig. 18.4 and illustrates the inflexible nature of text-based user interfaces where the software guides the user through the necessary user interactions. This diagram shows examples of using the *entry/* and *do/* actions within a state. The transition from the three named states each has a trigger that causes the transition to the next state.

The class diagram in Fig. 18.5 shows the changes to the ABA to implement the menu and guided prompts text-based user interface described above. The ABA_contactData class continues to have a relationship with all three MVC components, while the new ABA_A_viewRequest class only has a relationship with the view and controller components. The ABA_A_viewRequest class contains an enumerated type (REQUEST) which identifies the type of request the user has initiated

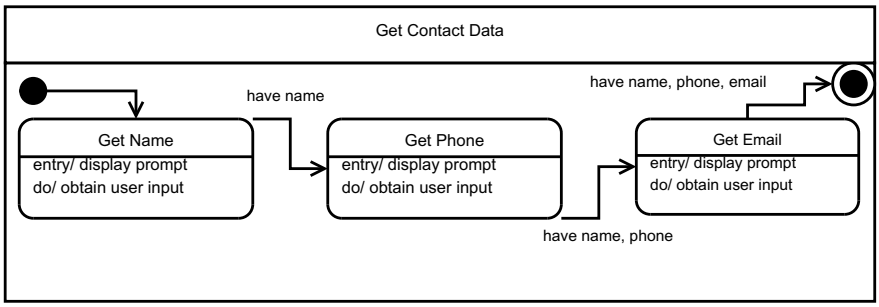

Fig. 18.4 State machine diagram—ABA Get Contact Data

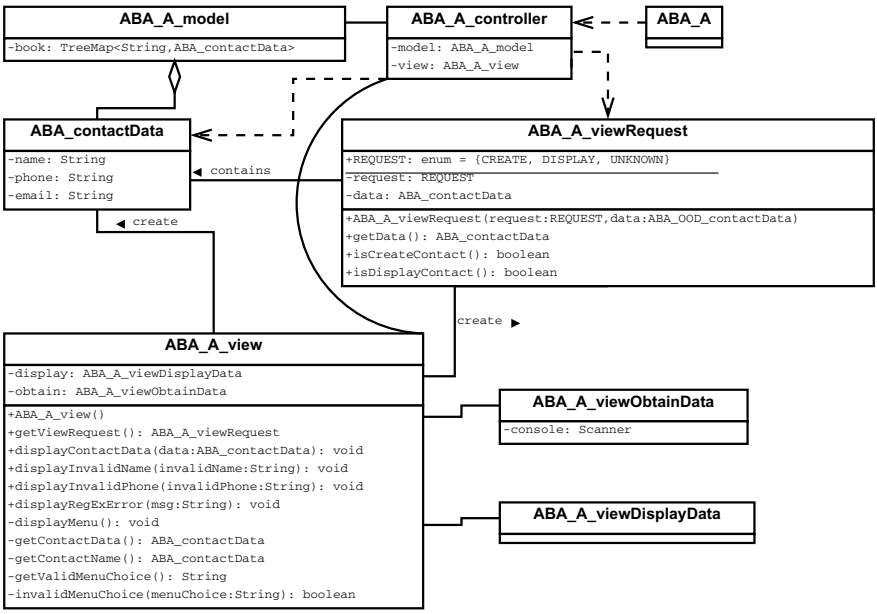

Fig. 18.5 Class diagram—ABA OOD Version A (menu and guided prompts)

via their menu selection. The getViewRequest method in the ABA_A_view class returns null to indicate the user has requested exiting the ABA.

As shown in the communication diagram in Fig. 18.6, the go() controller method now calls the getViewRequest() view method to get the user request and associated data. The getViewRequest() method displays the menu to allow the user to create a contact, display contact data, or exit the application. A ABA_OOD_A_viewRequest object is returned to the controller to indicate whether the user has requested creating a contact (request instance variable is REQUEST.CREATE) or displaying a contact (request instance variable is REQUEST.DISPLAY).

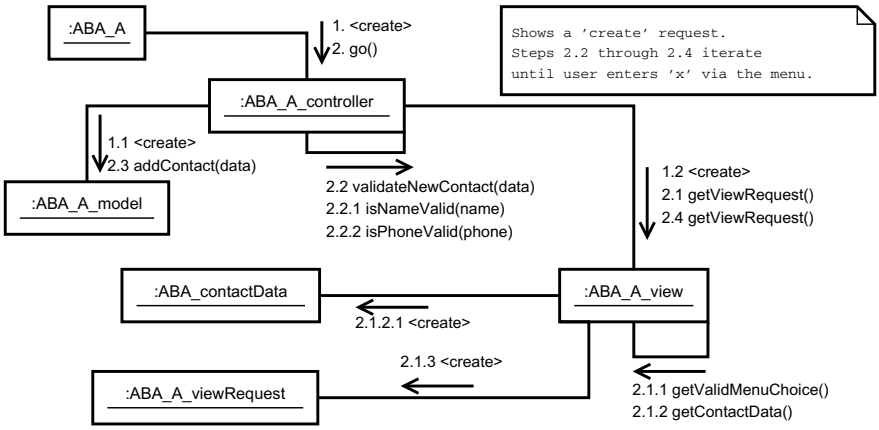

Fig. 18.6 Communication diagram—ABA OOD Version A (menu and guided prompts)

Figure 18.6 shows a create request, where the data instance variable in the ABA_A_viewRequest object contains the name, phone, and email values entered by the user. This data is first validated by the controller, and then if valid, given to the addContact model method to add this data to the address book. For a display request (not shown in the communication diagram), the data instance variable contains the name entered by the user. The name is first validated by the controller, and if valid, given to the getContact model method to obtain the contact data for this name. The ABA_contactData object returned by the model is then given to the view to be displayed.

18.5 OOD Top-Down Design Perspective

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

18.5.1 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.

18.5.2 OOD Personal Finances: TUI Design Choices

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. The number of user actions is large enough to warrant consideration of each text-based interface style discussed above.

A menu-based approach to providing a user with options seems like a reasonable approach. Table 18.4 shows two options for how the menu(s) could be designed. Option 1 shows a rather large menu listing all of the user actions. The user would enter a number (1 through 12) and then guided prompts would be used to obtain the data necessary for the option chosen. Option 2 has a much smaller main menu with just three options. Once the user enters a number (1 through 3), an appropriate submenu is displayed. The table shows the submenu for account management. Similar submenus would be displayed for transaction and report management. For this second menu option, efficiency could be improved by allowing the user to enter both the main menu and submenu choice at the `Enter choice:` prompt for the main menu. For example, guided prompts for updating a transaction would result from entering 2.3 at the main menu prompt.

A simple command-based user interface is shown in Listing 18.4. Having six commands seems like a reasonable design, one that could be learned fairly quickly. Four of these commands—create, delete, update, and view—would require a keyword be entered immediately after the command. The valid keywords would be account, transaction, and report to indicate which type of personal finances data entity is being referenced by the command.

Listing 18.4 Sample Personal Finances Commands and Data Entry Prompts

```
command> help
All commands are case insensitive
Enter 'C' or 'create' to create an account, transaction, or report
Enter 'D' or 'delete' to delete an account, transaction, or report
Enter 'H' or 'help' to display this help text
Enter 'U' or 'update' to update an account, transaction, or report
Enter 'V' or 'view' to view an account, transaction, or report
Enter 'X' or 'exit' to exit

command> C account
Enter account name: MyBank
Enter account type: savings
Enter starting balance: 100.00

command> delete transaction
Enter account name and type: MyBank credit-card
Enter transaction date: 03/23/2019
Enter transaction amount: 10.99
Do you want to delete the following transaction?
Date: 03/23/2019 Description: lunch Debit: 10.99
Enter Y or N: Y

command> X
```

Table 18.4 Personal finances menu design options

Menu Option 1	Menu Option 2 with submenu
1. Create account	1. Account management
2. View account	2. Transaction management
3. Update account	3. Report management
4. Delete account	X. Exit
5. Create transaction	Enter choice:
6. View transaction	
7. Update transaction	
8. Delete transaction	1. Create account
9. Create report	2. View account
10. View report	3. Update account
11. Update report	4. Delete account
12. Delete report	X. Exit
Enter choice:	Enter choice:

An example of a more complex command-based user interface is shown in Listing 18.5. The *KEY-VALUE-PAIR* shown in the help text represents a replacement for the

Table 18.5 Personal finances TUI design comparison

TUI Design	Efficiency	Learnability	Utility
Menu & Guided prompts	Maybe	Easy	Yes
Commands & Guided prompts	Yes	Moderate	Yes
Commands revisited	Yes	Hard	Yes

guided prompts used in the first two user interface designs described above. As shown in the two "create" examples, a KEY-VALUE-PAIR consists of a keyword (e.g., name, type, balance, account, desc, and debit) following by a colon and a data value (e.g., MyBank, savings, 100.00, lunch, and 10.99). This expanded version of a command-line interface would be much harder to learn when compared to the first two approaches.

Listing 18.5 Sample Personal Finances Commands Revisited

```
command> help create
All commands are case insensitive.
Command parameters in [square brackets] are optional.
Command parameters in {squiggly brackets} represent a grouping.
A grouping may be repeated using ellipses (i.e., ...).
Use single (') or double quotes (") to enclose data with spaces.
To create a data entity: c TYPE {KEY-VALUE-PAIR} ...
                         create TYPE {KEY-VALUE-PAIR} ...

command> C account name:MyBank type:savings balance:100.00

command> C transaction account:MyBank type:credit-card
         desc:lunch debit:10.99

command> X
```

18.5.3 OOD Personal Finances: Evaluate TUI Design Choices

Table 18.5 compares the three text-based UI design choices just presented. Assuming that the second option for the menu design is used, the ability to enter a number representing the choice from the main menu and submenu in one entry (e.g., 2.3) makes the menu design as efficient as the simple command-line option. The more complex command structure described in Listing 18.5 would be much harder for someone to learn, especially a user that occasionally uses the personal finances application.

Given everything discussed regarding a text-based user interface for personal finances, it seems reasonable to provide both the menu option 2 and simple command interfaces. Both of these approaches would use the same guided prompt design while allowing an occasional user the menu option 2 interface to efficiently use the

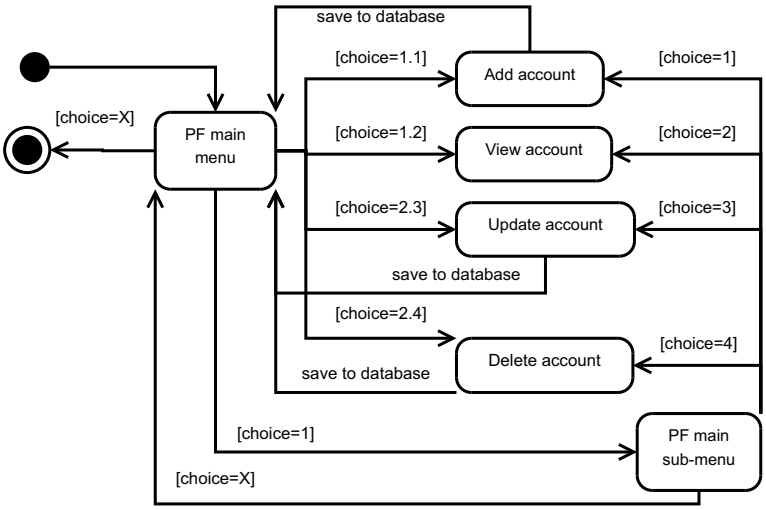

Fig. 18.7 State machine diagram—PF user interactions

application. While someone may argue that the more complex command structure as described in Listing 18.5 is the best choice for a power user, this user interface design includes design and implementation logic that is not usable by the other two approaches. Specifically, the parsing of a command must include logic for the KEY-VALUE-PAIR. This is unique to this approach. Combine this fact with learnability being hard for this design, and the conclusion drawn by the author is that the extra effort to design, build, and test the third design option is not worth the investment.

18.5.4 OOD Personal Finances: TUI Design Details

The statechart and UML class diagram in Figs. 18.7 and 18.8 show the modifications to the view component as a result of selecting a user interface design for the personal finances application.

The statechart shows the user interactions with the menu and submenu for an account. Similar behavior would exist for user interactions related to transactions and reports. Note the shortcut the user may elect to enter to get from the main menu directly to the guided prompts for adding, viewing, updating, or deleting an account. Entering *X* from the main menu would cause the personal finances application to end, while entering *X* from the submenu redisplays the main menu.

The UML class diagram contains a few significant changes when compared to the Chap. 15 class diagram shown in Fig. 15.15. First, an abstract class named viewUserData has been added to the design. Also note the inheritance relationships between viewAccount, viewTransaction, viewLabel, and viewReport and this new abstract class. This allows the view component to return user data related to any of these four data entities. This can be seen in the second significant change to the class

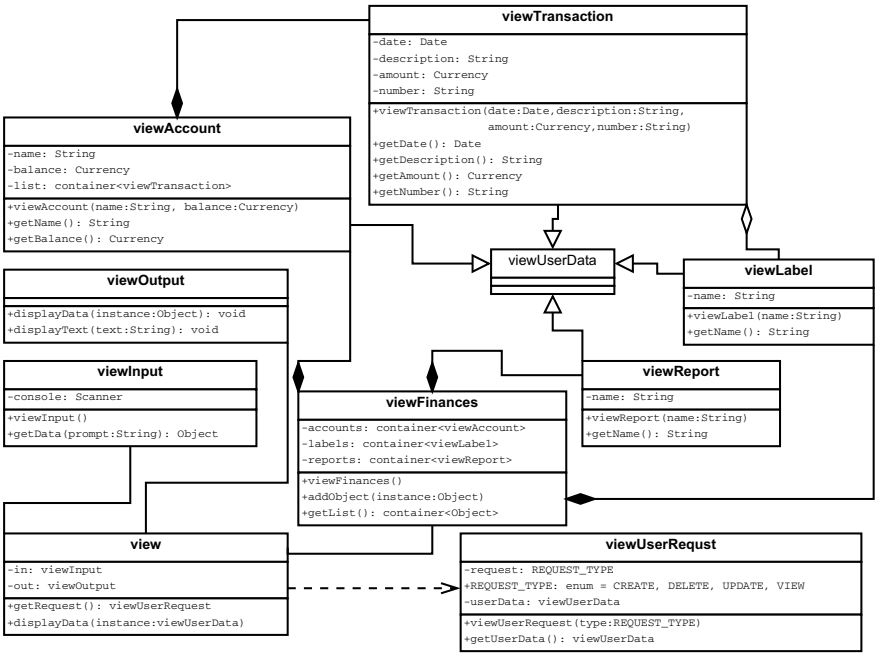

Fig. 18.8 Class diagram—PF view component

diagram, notably the getRequest public method in the view class. This method returns a viewUserRequest object that contains a viewUserData object. Along with the REQUEST_TYPE enumerated type defined in the viewUserRequest class, this allows the view to return a user request for creating, deleting, updating, or viewing either an account, transaction, label, or report.

18.6 Post-conditions

The following should have been learned when completing this chapter.

- A text-based UI should consider some combination of menus, commands, and guided prompts to provide a user with an efficient, useful, and easy to learn user interface.
- Using more complex commands in a TUI may be appropriate when many of your users are considered power users. Knowing your users is critical in determining whether this is a viable design option.
- 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

Discussion Questions

1. An ATM may not use a touch screen, instead it has buttons that get pushed to activate an option and uses a keypad to enter numbers. What type of text-based UI is this?
2. Can you think of any devices that use a text-based UI? If yes, what is the device and how would you describe this UI in terms of using prompts, menus, and/or commands? Evaluate this text-based UI using the HCI design criteria. Do you have any suggestions for improving this HCI design?

Hands-on Exercises

1. Use an existing code solution that you've developed, develop alternative text-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?
2. Use your development of an application that you started in Chap. 3 for this exercise. Modify your HCI design to use some combination of prompts, menus, and commands, and then evaluate your design using the HCI design criteria.

3. Continue hands-on exercise 3 from Chap. 12 by developing an HCI design using some combination of prompts, menus, and commands. 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