Appendix 6 - Projects

Overview

This appendix contains descriptions of many Smalltalk projects. Some are very detailed, others are just ideas that require some creative thought to arrive at a detailed specification. Some of the projects are easy, others are difficult.

A.6.1 1. When delivering a VisualWorks product to a customer, you might want to make it very difficult for the client to make sense out of your code - for competitive reasons. One strategy that has been suggested is to change the names of all your classes and methods to some meaningless sequences of characters. Develop a new class called Srambler which will do some or all of the following with each of the user-specified classes: Delete all comments (class and methods), rename the class and all its categories and methods to some random characters of strings. Note that the names must be changed wherever they are used, at all places in the library. Implementing this problem in a fully automated way is impossible because of polymorphism. You will thus have to implement the class so that the user provides some interactive help in the scrambling process.

2. **4.5 Dental Office**

4.5.1. System Description

We are to implement a simple implementation of software required by a small dental office. The program will keep records on individual patients including their bills, the schedule of patient appointments, information on staff including their schedules and paychecks, and keep track of dental material and potential suppliers. In more detail, the information to be kept includes the following:

- Patient record
    - Name (first, middle, initial)
    - Address (mailbox, street, city, and postal code)
    - Phone number
    - List of appointments (date, time, task, bill, payment received)
    - Dental record (tooth-by-tooth: filled, extracted, nothing)
    - Amount currently due
    - Notes
- Staff record
    - Name (first, middle, initial)
    - Address (mailbox, street, city, and postal code)
    - Phone number
    - List of shifts (dates)
    - Hourly wage
- Material
    - Item name
    - List of suppliers
- Supplier
    - Name of company
    - Address
    - Phone
    - List of materials supplied with prices

We want to be able to perform the following tasks:

- Patient record
  - Add
  - Delete
  - Display
  - Edit
  - Print
  - Create appointment
  - Cancel appointment
  - Change appointment
  - Print information about appointment
- Staff
  - Add
  - Delete
  - Display
  - Edit
  - Print
  - Schedule (assign, change, cancel)
  - View shifts for current month
  - Print paycheck
- Material
  - Add
  - Delete
  - Display
  - Edit
  - Print
  - View all potential suppliers
  - Create order
  - Print order
- Supplier
  - Add
  - Delete
  - Display
  - Edit
  - Print

The Context Diagram is as in Figure 4.8.

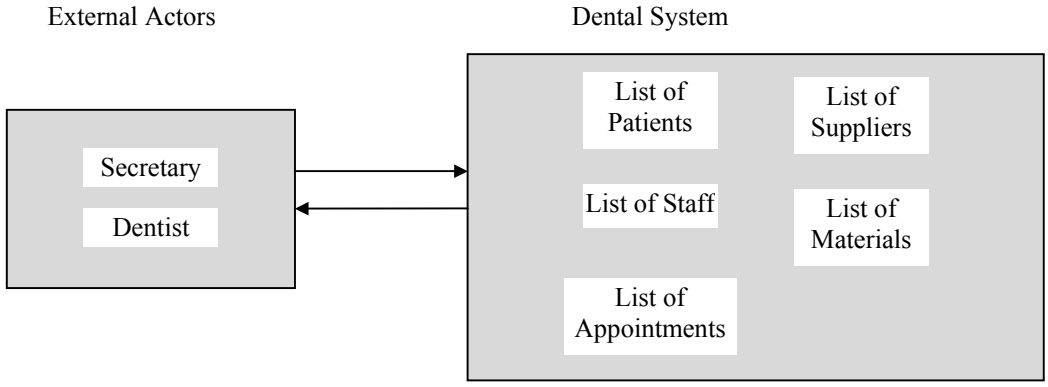

Figure 4.8. Context Diagram of Dental Office.

Our next task is to find the scenaria. Because this system is much more complicated than the library catalog, there will be many scenaria and we will thus restrict ourselves to a few examples only.

Scenario 1: Adding a new patient
The operator (secretary or dentist) selects the *New patient* task from the main window of the user interface, the system opens a window with fields for all required patient information, the operator enters all required information and clicks *Accept*, the system stores the new patient record.

Scenario 2: Making an appointment
The operator selects the patient from the list and the *Open patient record* task in the main window, the system opens a window with the patient record, the operator selects the *Appointment* task, the system opens an appointment window, the operator fills in the information, clicks *Accept* in the appointment window which closes, clicks *Accept* in the Patient record window which closes, the system stores the information.

Scenario 3: Creating and printing a material purchase order
The operator selects the *Create purchase order* task in the main window, the system opens a window with a list of materials, the operator selects a material, the system displays a list of suppliers of the selected material, the operator selects one, completes the *Quantity* field, clicks *Accept and Print*, the system saves the order and prints it.

This time, we will only show the main user interface window and leave it to you to decide which other windows are required and to design them. The main window is shown in Figure 4.9.

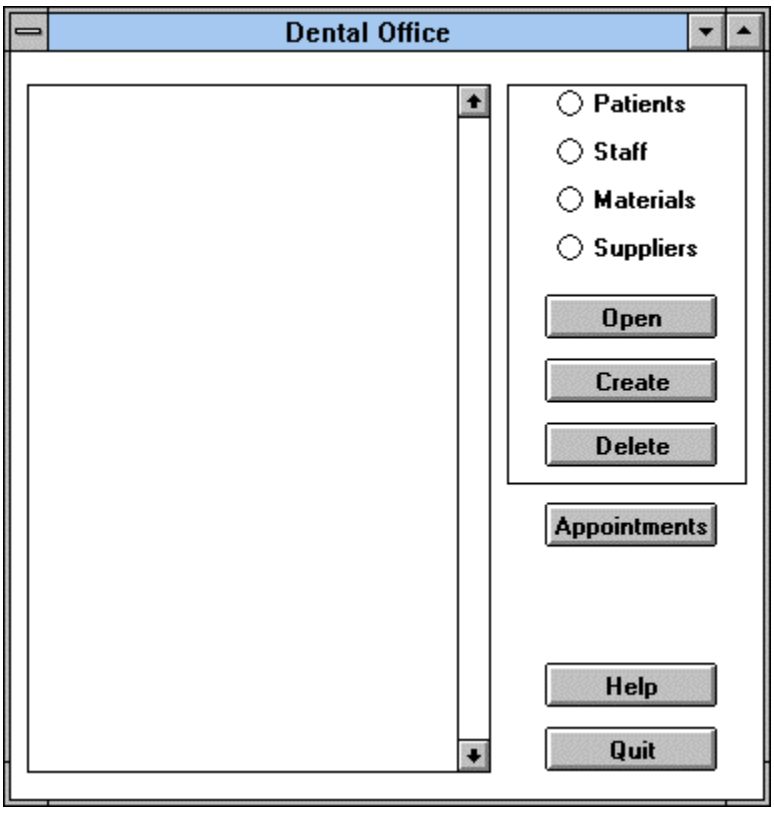

Figure 4.9. Main window of Dental Office.

3.

4.3 Computerized Library Catalog

4.3.1 System Description

We are to develop a computerized catalog for keeping track of books and borrowers. The program will allow authorized personnel (referred to as librarians) to perform the following tasks:

- Book
    - Add a new book to the catalog
    - Modify an existing book
    - Delete a book
    - Sign a book out to a borrower
    - Sign a returned book in
    - Search for a book in the catalog
- Borrower
    - Add a new borrower
    - Edit information about an existing borrower
    - Delete a borrower
    - Search for a borrower

The information held about books and records is as follows:

- Book
    - One or more authors identified by first, middle, and last names
    - Title
    - Publisher name
    - Year of publication
    - In or out of library
- Borrower
    - First name, middle name, and last name
    - Address consisting of street name and number, city, telephone number
    - List of borrowed books with dates when the books were borrowed

The specification assumes the user interfaces shown in Figures 4.5, 4.6, and 4.7.

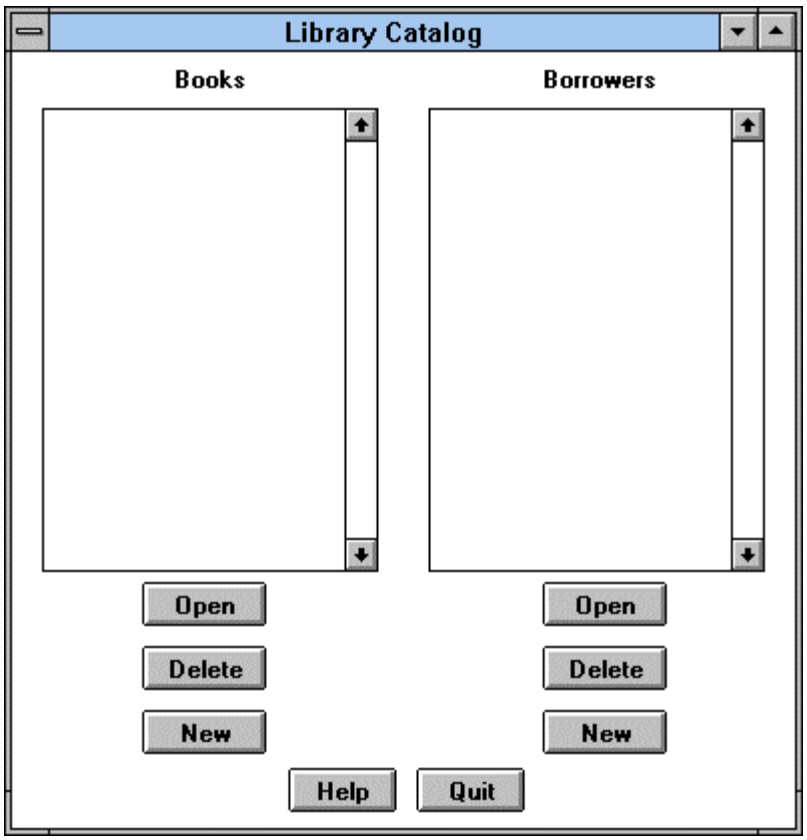

Figure 4.5. Main window of library catalog.

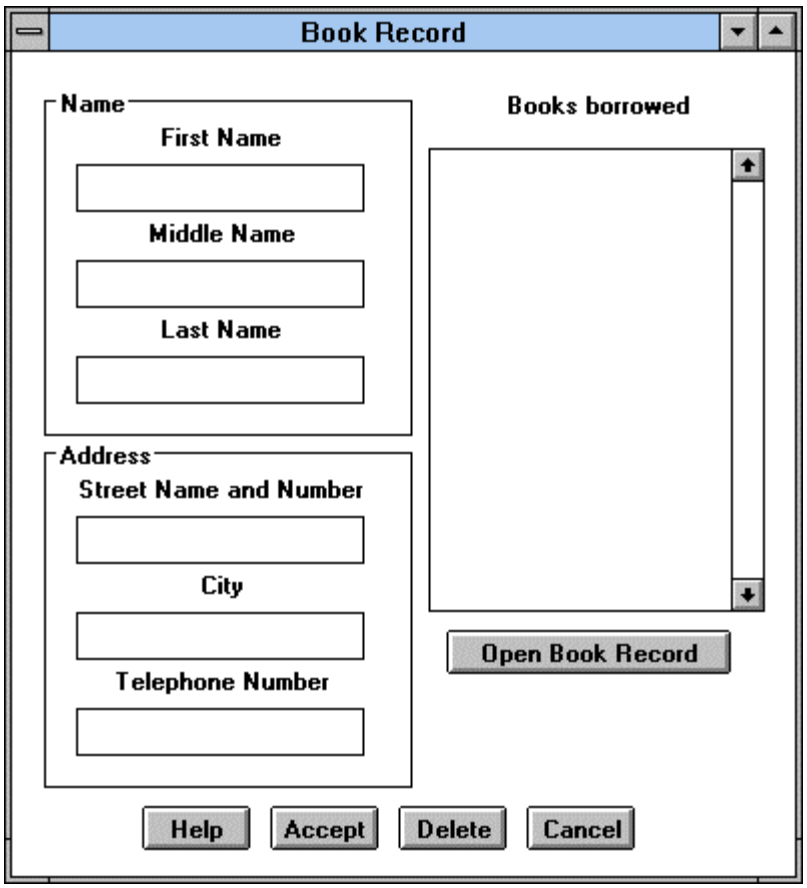

Figure 4.6. Book information window.

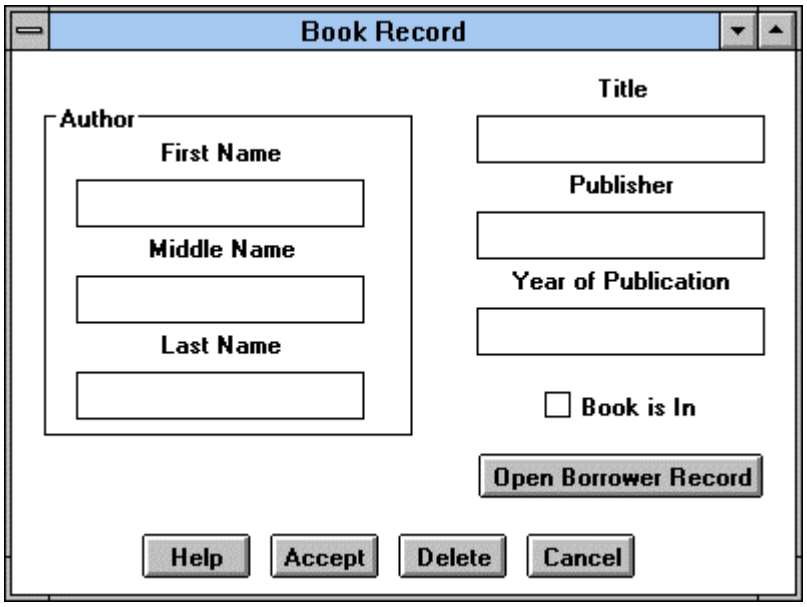

Figure 4.7. Borrower information window.

The following are some of the main usage scenaria:

Scenario 1: *Librarian opens the application.*
High level conversation:
1. *User* types a Smalltalk expression and executes it.
2. *System* opens a password window
3. *User* types password.
4. If the correct password is entered, *System* opens user interface shown in Figure 4.1; a notifier window opens otherwise.

Scenario 2: *Librarian adds a new book.*
High level conversation:
1. *User* clicks *New book* in the main window.
2. *System* opens a form window providing text fields for all required information. We will call this window the book form.
3. *User* enters at least author name and book title and indicates end of task by clicking *Accept* in the form window.
4. *System* closes form window and updates book catalog.

Scenario 3: *Librarian modifies an existing book record..*
High level conversation:
1. *User* selects the desired book in the catalog and clicks *Open*.
2. *System* opens a book form with current information about the book.
3. *User* edits the form and indicates end of task by clicking *Accept* in.
4. *System* closes form window and updates book catalog.

Scenario 4: *Librarian deletes a book.*
High level conversation:
1. *User* selects the desired book in the catalog and clicks *Delete*.
2. *System* opens window requesting confirmation that the book is to be deleted.
3. *System* closes confirmation window and updates book catalog if the user confirmed.

Scenario 5: *Librarian signs a book out to a borrower.*
1. *User* selects the desired book in the catalog.
2. *User* selects a borrower.
3. *User* clicks *Borrow book.*
4. *System* updates book and borrower catalogs.

Scenario 6: *Librarian signs a book in.*
High level conversation:
1. *User* selects a book.
2. *User* selects a borrower.
3. *User* clicks *Return book*.
4. *System* updates book and borrower information.

Scenario 7: *Librarian adds a new borrower.*
High level conversation:
1. *User* clicks *New borrower* in the main window.
2. *System* opens a form window providing text fields for all required information. We will call this window the borrower form.
3. *User* enters at least borrower name and address and indicates end of task by clicking *Accept* in the form window.
4. *System* closes form window and updates borrower catalog.

Scenario 8: *Librarian modifies an existing borrower record.*

High level conversation:
1. *User* selects borrower in the catalog and clicks *Open*.
2. *System* opens a book form with current information about the book.
3. *User* edits the form and indicates end of task by clicking *Accept* in.
4. *System* closes form window and updates borrower catalog.

Scenario 9: *Librarian deletes a borrower.*
High level conversation:
1. *User* selects borrower and clicks *Delete*.
2. *System* opens window requesting confirmation that the borrower is to be deleted.
3. *System* closes confirmation window and updates borrower catalog if the user confirmed.

Scenario 10: *Librarian requests help information.*
High level conversation:
- *User* clicks *Help*.
- *System* opens help window.
- *User* clicks *Close* in the help window.
- *System* closes help window.

This scenario is so obvious that it is questionable whether it should be included. We will not include such trivial scenaria in the future.
A variety of other important scenaria exist and we leave them as exercises.

The specification and scenaria that we just read contain several new terms and it will be useful to create a dictionary with their exact definitions. *We will add a Term Dictionary as a new deliverable for Specification of Requirements.* The dictionary for our problem is as follows:

- *Book form* - dialog window in Figure 4.6. Contains book information and additional control buttons.
- *Borrower form* - dialog window in Figure 4.7. Contains borrower information and additional control buttons.

(A dialog window is a window that must be completed and closed before another window can be accessed.)

Context Diagram

The last task is to draw a diagram showing the main components and deciding which of them are a part of the application, and which external actors. From the description, we conclude that there is a single external actor - the librarian - and that the system to be developed consists of a user interface, an application model, a catalog of books, and a catalog of borrowers.

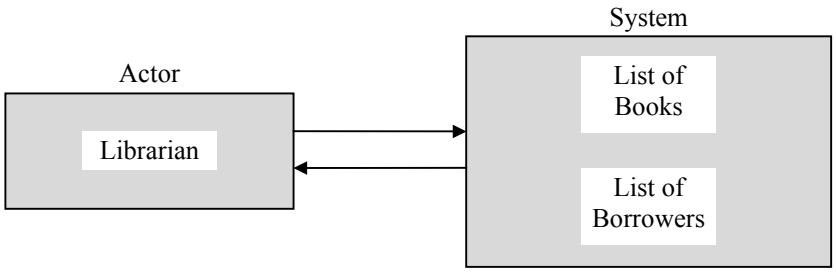

Figure 4.4. Context Diagram showing readily identifiable parts of Library Catalog and external actors.

This completes System Description and we proceed to Preliminary design.

4.3.2 Preliminary design

The steps in Preliminary Design are

1. Identify candidate classes and create CRC cards with their brief descriptions.
2. Identify responsibilities of candidate classes and record them on their CRC cards.
3. For each class, identify collaborator classes required to fulfill this class's behaviors and record them on the CRC card.
4. Verify completeness of CRC cards by walkthroughs, checking that the identified classes and their behaviors can implement all scenaria obtained during the System Description step.

Identify candidate classes and their responsibilities

This is the most difficult part of the whole procedure and the commonly proposed guideline for finding the classes is to read the description carefully, looking for nouns as possible class candidates. In our relatively simple example, the obvious candidates are the windows and the other components of the user interface, the list of books, and the list of borrowers.

One would expect that *windows and their components* are already in the Smalltalk library and this is indeed the case. We can thus eliminate them from our consideration since we are only interested in the *new* classes that we will have to create.

The list of books and the list of borrowers are both lists, collections of objects. Collections of objects appear in almost every application and we would again expect that suitable collection classes are already in the Smalltalk library. This is indeed so and we thus don't have to include them in our design.

The only remaining objects therefore are books and borrowers. A book object is basically an information object and its intelligence is limited to knowing certain information, being able to provide it when requested (for example when a window needs to display it), and being able to change it when requested (when the librarian enters new information). A borrower object is another information object capable of holding and accessing information; only the type of information it holds is different. A reasonable conclusion is that we need to declare two new classes - *Book* and *Borrower*. This, however, probably is not the best solution for two reasons: The first is that both of these objects contain a bit too much information, the second is that the information held by both *Book* and *Borrower* contains smaller and logically self-contained concepts - person information and address information - and that some of this information - namely person information - is, in fact, shared by both. We conclude that a more elegant solution is to define the following classes:

Address: Contains and can access information about street name and number and city.
Book: Contains and can access information about book consisting of author, title, publisher, and isIn information.
Borrower: Contains and can access information about borrower consisting of borrower, address, telephone number, and list of borrowed books.
Person: Contains and can access the first, middle, and last name of a person.

Besides greater simplicity of these classes, the advantage of this approach is that we may be able to reuse them in another application. After all, persons and addresses are very common. We summarize our decisions on CRC cards as in

Book: I hold information about a book in the library - its author, title, publisher, year of publication, and status (in or out).

Responsibilities	*Collaborators*
Get/return author information	Person
Get/return title information	
Get/return publisher information	
Get/return year information	
Get/return isIn information	

We leave it to you to complete CRC cards for the remaining classes as an exercise.

Verify completeness of class list by scenaria walk throughs

Scenario 1: *Librarian opens the application.*
The librarian types a Smalltalk expression and executes it. The application open the user interface

Scenario 2: *Librarian adds a new book to the catalog.*
Librarian selects the *New* task from the main window, a window appears providing text fields for all required information. The librarian enters at least author name and book title, indicates end of task by clicking *Accept* in the main window, the application closes the window and updates the internal catalog.

Assuming that we have a working user interface, clicking the appropriate buttons and entering information stores the information in the object associated with the interface. Clicking the Accept button then requests a new Book object from the Book class, transfers the collected information to it, and adds it to the book catalog object. The scenario thus seems executable. However, there is a small gap here - we have a Book class and the Person class defining its major component, we also have classes implementing windows and their 'widgets' (buttons, labels, lists, input fields, and so on) - but we don't have a class to tie the user interface to the book list, borrower list, book objects, and borrower objects. In Smalltalk terminology, we have the UI (user interface) classes and the *domain* classes describing the objects in our problem domain, but we don't have the *application model* (Figure 4.5).

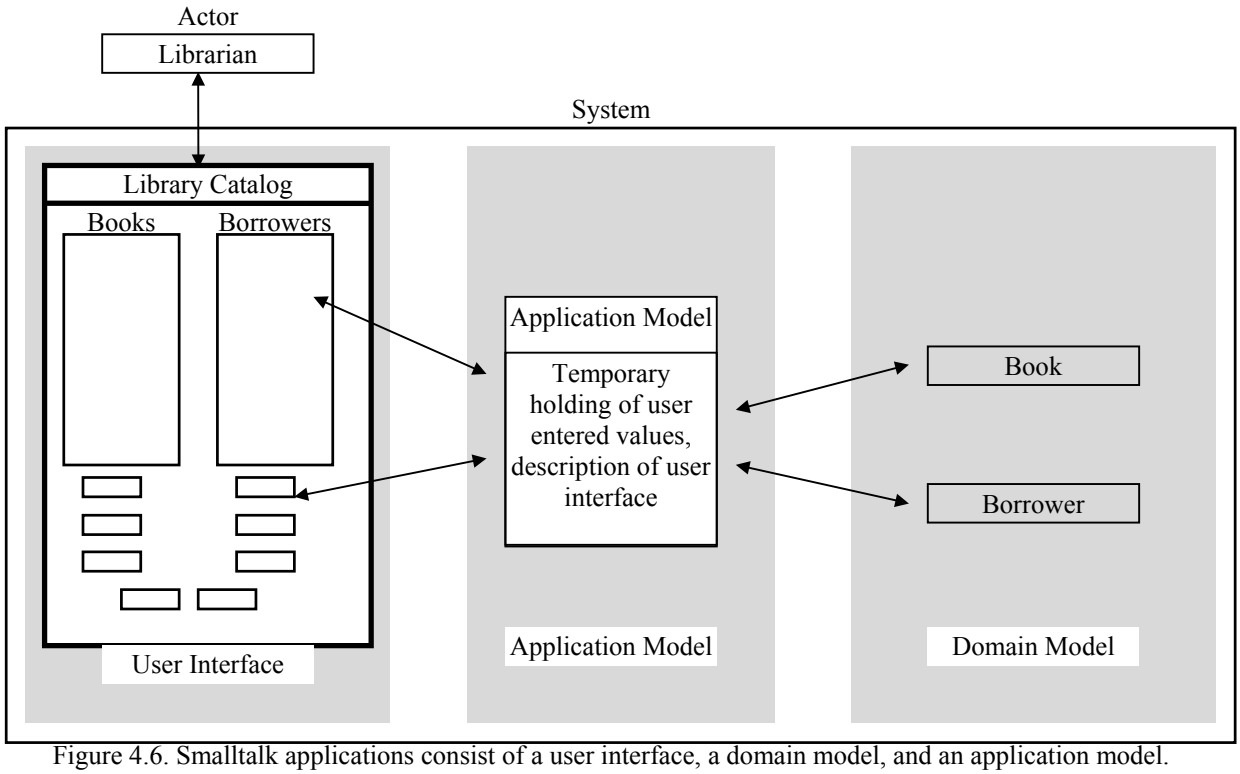

Figure 4.6. Smalltalk applications consist of a user interface, a domain model, and an application model.

We will thus add another class called LibraryCatalog. Its responsibilities will include

- holding the description of the user interface (how the window and UI widget objects and laid out and how they communicate the information that they display or gather)
- holding information entered by the librarian before it is accepted and transferred to domain objects.

We leave it to you to verify that our classes can handle all the remaining scenaria.

4.3.3 Design Refinement

In this step, we must decide whether some of our classes are sufficiently similar in their behavior to warrant declaring a superclass with the shared behavior, and we must provide details for the behaviors that we identified in the previous step. In our case, the classes that we identified don't share any behaviors and there is no justification for any new superclasses. All classes, with the exception of LibraryCatalog have no shared behavior with any existing classes in the Smalltalk library and we will thus define them as subclasses of Object. Class LibraryCatalog, on the other hand, contains mainly behavior that is shared by all application models - it holds the description of the user interface and provides access to domain objects - and it is not surprising that VisualWorks contains an abstract class providing much of this behavior. This class is called ApplicationModel and LibraryCatalog will thus be a subclass of ApplicationModel (Figure 4.7).

Class name: LibraryCatalog	Superclass: ApplicationModel
Responsibilities	Collaborator classes
Hold description of user interface	
Hold book information until accepted	
Hold borrower information until accepted	

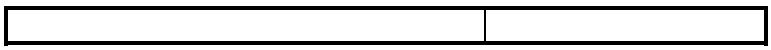

Figure 4.7. CRC card of class LibraryCatalog.

The last step is to decide whether we should refine some of the responsibilities that we identified in the previous steps. Since our domain classes are trivial information objects, all their simple behaviors are accessing behaviors that don't require any elaboration. Since we cannot implement our classes, we are thus finished.

Example: Puzzle-a-day program

use sorted dictionary but implement it as sorted collection.

Chapter projects

All project classes are available in the file-in code for this book.

Project 1: Function calculator

Re-implement Project 1 from Chapter 5 using the user interface in Figure 3.30. The project is available as Project4 in the fileout.

Figure 3.30. User interface for Project 1.

Project 2: The Hangman game?

Implement the Hangman game with the interface in Figure 3.31. Use the Divider and Region widgets from the canvas, beVisible and beInvisible (?) messages. Explain game. Read Chapter Input Fields, etc. first

Project for Chapter 8

Implement the bank simulation project used as an example in Chapter 2.

Projects for chapter 13

1.
2. Elevators
3. Port
4. Bank

1. Extend the shopping minder from Chapter 9 into the following application: The user interface consists of a single window with two single-selection list widgets and several buttons. The list on the left contains all available items and their unit prices, the list on the right contains items that the user selected and their counts. The buttons include: *Add* - asks the user for a new item name and adds it to the list of available items or increments the item's count if it is already in the list, *Get* - adds the item selected in the left list and adds it to the right list, *Delete* - subtracts one from the number of occurrences of the item selected in the right list, *Print* - prints information in the right list in the Transcript, *Close* - closes the application.