

Project: applet-based photo album (and more)

March 20, 2007

New - 20 March 2007 — homework 7, due Wednesday 28 March

Introduction

Summary

This team project builds software to compose and output an album of pictures and text. The project exercises concepts from course material (object oriented design, Java features) and also some tools for cooperative software development.

Album Concept

The album software has two basic tools that are visible to customers (users), namely the *composer*, which allows users to create albums from a library of photos, and the *player*, which displays an album. The composer has extra functionality beyond just album creation: using the the composer, one can edit an existing album, so that the user can rearrange, add pictures and text, remove pages, and other typical editing functions. The player also has different functions. In slideshow mode, the albums pages turn automatically; in interactive mode, the player responds to user requests to go forward and back. If this were to be a commercial product, one could imagine that the player is free, but the composer has to be purchased (which is why these two are not bundled into just one program). In addition to the basic features listed in this paragraph, there can be many other ways to jazz up both composer and player; one could imagine, for instance, that the player also has support for outputting the album to high-resolution color printing devices. Beyond player and composer, other useful programs could be for photo library management, running test suites, and converting between one album format to another.

Album and Software Structure

An album contains *pages*, and each page contains photos, text, and possibly other objects. The photos and text have various attributes (place on page, etc). It is vital for this project that each team have a *fallback design*, which is a basic and rudimentary design that (we hope) is easy to implement. Your team's first priority should be this fallback design, in which the album is quite simple (and boring) but working.

Beyond this basic design, you can also have a more interesting, feature-rich design and implementation. For instance, in a more advanced implementation, maybe pictures fade in, animate, and text similarly can do interesting things. An album could have an index, with thumbnails, have search functions, and the like. These things may require substantial effort, and because this is a team project, you could be sure about the ballback design before trying the fancier things.

Software for Development

Java's many libraries support development of composer and player. We want you to use *applets* for the interactive programs; Eclipse has applet support built in, and there is also a command-line way to run an applet (called the appletviewer). You can even run applets from a web browser, though to allow anyone to use your applet over the web, you would need access to a web server. Java also gives you library functions for reading and writing files, converting objects to text streams, ways to write test suites, and so forth. In support of team software development, we have Eclipse and its plugins for CVS (Concurrent Versions System). Your team will use CVS for sharing code and other documents between team members, and also for showing instructors your team's progress.

Team Roles

We've identified four roles for team development; three of these are principal roles; the fourth is a testing or validation role. The three principal roles correspond to different aspects of the project.

Role 1 is the design and implementation of the composer's GUI. The activities here require the planning of how users will interact with the composer, analysis of what the composer needs in terms of object storage, functions and operations on objects, and some study of how applets work in Java.

Role 2 is the design and implementation of the player's GUI. As with Role 1, the designer needs to describe the player's features in some detail and think through how these features would be programmed using classes, objects, and other functions.

Role 3 is the design and implementation of the *backend* (the term “backend” is industry-standard jargon for software that manages data, but doesn’t have a GUI component). The designer in this role needs to plan the classes and objects that will be used by the composer and player, study the issue of *persistence* of objects, and also master operations for reading and writing to photo libraries and album files.

The fourth role, the validation role, consists of working with other team members to understand the requirements and then write programs and test suites to make sure the code of the other members is correct. This is especially important in production software development, where typically software is implemented in *cycles* (version 0, version 1, etc); with each new version, it is helpful to automate the process of testing and debugging so that new versions don’t create new bugs.

Role Interaction

The three principal roles interact more than you might initially suspect. A Role 1 implementor is *not allowed* to design and implement objects and methods for reading from files — that is the job of Role 3. Also, Role 1 shouldn’t be implementing player functions, since that would duplicate effort of Role 2. The backend is crucial to the other roles, supplying classes and methods for all the album’s objects; but Role 3 cannot design everything in isolation, because it’s up to Roles 1 & 2 to say what objects are needed and what kind of methods should be provided for those objects. Role 1 and Role 2 are similar in that they are both applet-GUI functions, yet the composer has many interactions outside of what a player can do, and the player may have timing controls, a more attractive look and feel, and some versatility that the composer doesn’t need.

For any team, it is possible that Role x , for $x = 1, 2, 3$ does a better job than the other two. It could be that Role 1 delivers a fantastic composer, but that the player remains rather primitive; and it could be that Role 3’s contribution includes extra programs to analyze and summarize an album’s content in detail, regardless of what Role 1 and 2 have done.

So how to team members set up boundaries, decide on what conventions to use, and share code? First, each team member needs to think independently about the specific needs and concerns associated with the role. Second, we can use the modeling and documentation techniques of the textbook; use cases for the GUI programs, sequence diagrams to document detail of how users communicate with composer and player, and UML to figure out what kinds of objects are going to be used. The backend design can also use UML and sequence diagrams between program and files, and even a state diagram to model the backend’s behavior. At some point, the team should have a meeting to agree on names and basic (fallback) functions.

Once the team comes to basic agreement on the design, and has documented that using UML or other methods, the team starts to divide up the work to be done. Role 3 needs to provide Java *interfaces*, even before any code is written, that Roles 1 and 2 will use for

management of persistent objects. Role 2 may need to provide interfaces to Role 1, if the composer program calls up functions provided by the player. Later, when classes are designed and implemented to fulfill the designed interfaces, Java packages and code will need to be shared amongst the team members using CVS. Meanwhile, the course instructors may inspect the progress of the team and request clarification and refinement of the design, to make sure things are on track.

Homework 7

Due 28 March 2007. Part of the homework is on paper (for corrections and comments), part of the homework is online.

Part 1. Create a Java project, using Eclipse, that contains (i) a Java interface, (ii) a class that implements your interface, and (iii) a main method that uses this interface. We're not looking for anything fancy here, your project can be almost as simple as "Hello World", so long as it exercises an interface.

Turning in Part 1. Use CVS to turn in your Java project, following the procedure shown in class, but change the CVS repository location from

```
/group/class/c022/TeamSandBox to /group/class/c022/Homework7/yourlogin
```

where "yourlogin" is your login account on the department's cluster. If you didn't take notes, replay the lectures using the flash presentations found via

<http://www.cs.uiowa.edu/~herman/c022/>

(these flash presentations can be viewed on the department linux machines, using headphones to get the soundtrack, and only the first one or two presentations are useful for this homework).

Part 2. In brief, Part 2 of the homework is to prove that you've spent at least three hours thinking about the design questions for your team role (every student has been assigned one of the three roles; see Prof. Cremer if you don't have your role assignment yet).

So, how do you "prove" that you've carefully considered how to design the software for your team role? By the end of three hours of thinking, doodling, perhaps discussing with other members of your team (if you know who your team is), you should have some idea of the basic features you'll need to implement and how this can be programmed using Java. In fact, you should have:

- A list of the major features.

- For each feature, some description of how it works in terms of user interaction (Roles 1 & 2) or file interaction (Role 3).
- For Role 1 or Role 2, you should have some idea of what kind of methods and objects Role 3 is expected to deliver — you can write some description of what Role 3 should provide.
- For Role 3, you can imagine what kinds of services would be useful to Role 1 and Role 2, and then design classes and methods for those services.
- Possibly you can even write down the specific Java classes you will need to use (which are in Java libraries), and state what kind of techniques you'll need to learn. If you're going to create your own classes, interfaces, and such, you can list those.

In short, *if you have really thought through what it will take to do the job for your team role*, then you should have no trouble writing at least a couple of pages for Part 2 of this homework and turning that in on 28 March.

Notes:

1. Higher scores on this homework for using concepts from the textbook, such as UML, use-case scenarios, sequence and state diagrams.
2. Hang on to your homework – it will be useful for team meetings to have your analysis so you can productively talk about your ideas; better yet, you can even put your design ideas into CVS to share with your team.
3. The “three hours” mentioned above *does not include* the time spent writing up the document that your turn in; that may take an additional hour or so.