SYSTEM DESIGN DOCUMENT FOR DESKTOP HIPSTER

Version: Last iteration

Date: 2014-05-19

Author: Edvard Hübinette

This version overrides all previous versions.

1. INTRODUCTION

1.1. DESIGN GOALS

The design aims to be loosely coupled to enable freestanding view implementations to represent the backend. This way new graphical designs are easily added to an existing application base without the need to tinker with the original implementation design of the controller or the model. For usability, see RAD.

1.2. DEFINITIONS, ACRONYMS AND ABBREVIATIONS

- GUI, graphical user interface.
- Java, platform independent programming language.
- Client, the computer currently running the application.
- Host, an image host service on the web (e.g. Tumblr, Twitter, Imgur et cetera).
- API, Application Programming Interface. In this context it refers to web-APIs, the way out implementation talks to web servers.
- OAuth, a standardized protocol for authenticating to web services.
- MVC (model view control), A design pattern created to separate application logic, data and graphical representation in different packages. (Eckstein, 2007)
- Filter, a pre-made modification for applying to an image that can change how colors look, add textures etc.
- Serialization, the act of converting an object into a state that can be stored on disk, in this case in form of a file buffer. (Oracle, 2013)
- Tags, string identifications that can be applied to an image to make it part of a group for easier identification.

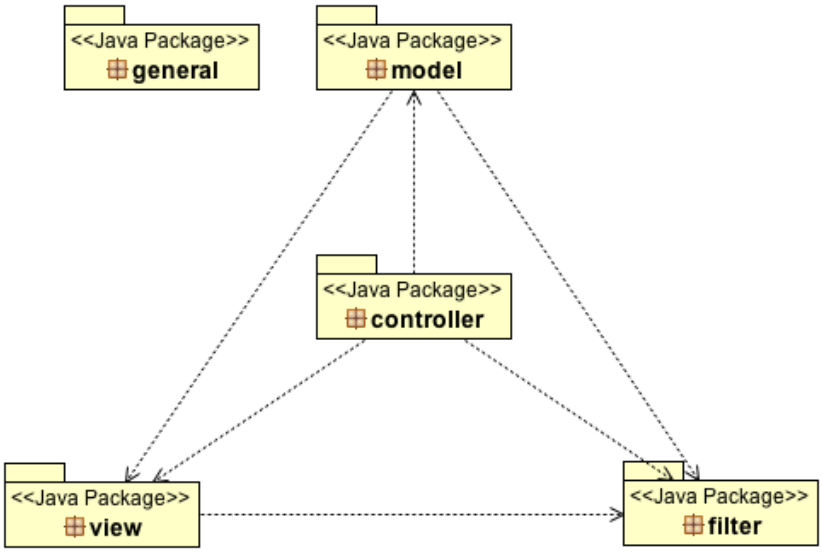
2. SYSTEM DESIGN

2.1. OVERVIEW

The application will use a modified MVC structure for managing separation of concerns in the implementation.

1.2.1. GENERAL

This is the project package structure with how they relate to each other. Detailed intra-package structure diagrams can be found in the appendix.



1.2.2. MODEL FUNCTIONALITY

The model is responsible for managing the library of our image representation as well as applying the actual image filters.

The library makes sure images does not disappear when the application is restarted by writing the database to disk as well as keeps track of all the edited versions of the imported images.

1.2.3. EVENT HANDLING

The vast majority of the communication that breaks the intra-package borders takes place through events. There are never more than two listeners and there is no need to detach/attach clients more than on initialization, so there was no immediate need for an event bus. The view makes decisions on model state change-events and communicates with the controller through events fired on user interaction.

2.2. SOFTWARE DECOMPOSITION

2.2.1. GENERAL

The application is decomposed in these modules.

- The view represents the model and application graphically.
- The model manages data.
- The controller picks up events and takes decisions on user interactions and translates these decisions to model method calls.
- The filter package contains the actual image-manipulation implementation (individual filters) and an enumerator that works as the usage interface.
- The general package contains the startup class and property name constants.

2.2.2. DECOMPOSITION INTO SUBSYSTEMS

Both filters and hosts are implemented in a stand-alone way to make it possible to be used freely by classes both in view, control and model packages. The sole contact point in both cases is an enumerator class.

Examples of why this is preferable: it fulfills the needs of the view to quickly render preview-edits for the user to get immediate modal feedback of changes without having to render the effects on the full-size original picture. It's also very practical for a view implementation to be able to access progress statistics directly from the web host if one would want to show such information to the end user.

Hosts are also implemented as single-user for convenience in development and grading; otherwise every run the user would have to log in to Twitter and/or Tumblr to authenticate. When running, Twitter uploads to https://twitter.com/ehubinette and Tumblr to http://ehubinette.tumblr.com.

2.2.3. NOTABLE DEVELOPMENT DIFFICULTIES

The filter implementation required a lot of research and testing to reach a fast and qualitative solution for editing colors, converting to black and white and, mostly, layering pictures over one another.

The implementation of hosts was difficult for certain services because of horribly documented APIs, notable Flickr that later on was removed from the project because of stability issues.

To get the application to play nicely with the OS X file system a lot of testing went into the Finder open/save dialog-integration and Drag and Drop functionality since the application only accepted certain file types and different actions had to be taken depending on which it was. The GUI also needed to inform the user immediately through modal feedback about whether the file was accepted or not.

The hardest issue to solve that was very high priority in the project was the serialization of the library. The ExtendedImage objects that store the image data, together with its current tags and different versions, aggregate instances of BufferedImage, which are not serilizable. Both plain and in maps, manual serialization had to be enabled to write the objects to disk and read them in at the next start. Since this has to be done with the same stream in different classes a very large amount of time where put into the development of this.

2.2.4. LAYERING

Layering is as indicated by Figure 1.

2.2.5. DEPENDENCY ANALYSIS

Dependencies are shown in Figure 1. There are no circular dependencies.

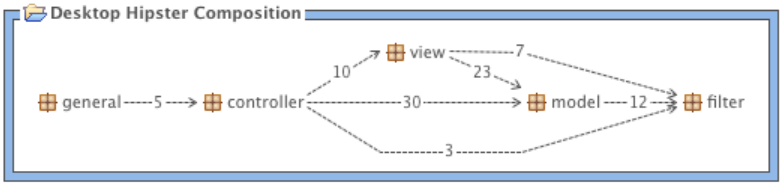


Figure 1: Layering and dependency analysis export from STAN

2.3. CONCURRENCY ISSUES

NA, this application is single threaded. Events are handled by the swing implementation. Threading the uploading procedure and library read-in could increase GUI responsibility and performance, but there has been trouble making the implementation of that play nicely with the serialization process in the save state-functionality.

2.4. PERSISTENT DATA MANAGEMENT

The applications current state, including imported images together with their respective edited versions and applied tags, as well as application-leveled fields like registered tags.

2.5. ACCESS CONTROL AND SECURITY

The application is in this release acting with hard-coded access tokens constants when communicating with web hosts. For using web-based authentication the application needs to be approved by Tumblr, Twitter and alike to get access to that authentication API.

The connection to the webhosts is run over https, an encrypted communication protocol, and authentication with the host APIs use OAuth. This is a very secure and standardized way of communicating securely; however the

local source code contains the access keys in clear text. This is sub-par from a security standpoint but can be seen as acceptable in a product on this level because of testing and grading since it is very convenient, there is really no security risk since the user that generated the keys can revoke them at any time and the application is used in a trusted environment.

2.6. BOUNDARY CONDITIONS

NA. Application is launched and exited as a normal OS X application.

3. REFERENCES

Eckstein, R. (2007, March). *Java SE Application Design With MVC*. (Oracle) Retrieved 04 03, 2014, from Oracle Technology Network: http://www.oracle.com/technetwork/articles/javase/index-142890.html

Oracle. (2013). *Interface Serilizable (Java Platform SE 7)*. Retrieved 04 12, 2014, from Oracle Web site: http://docs.oracle.com/javase/7/docs/api/java/io/Serializable.html

4. APPENDIX

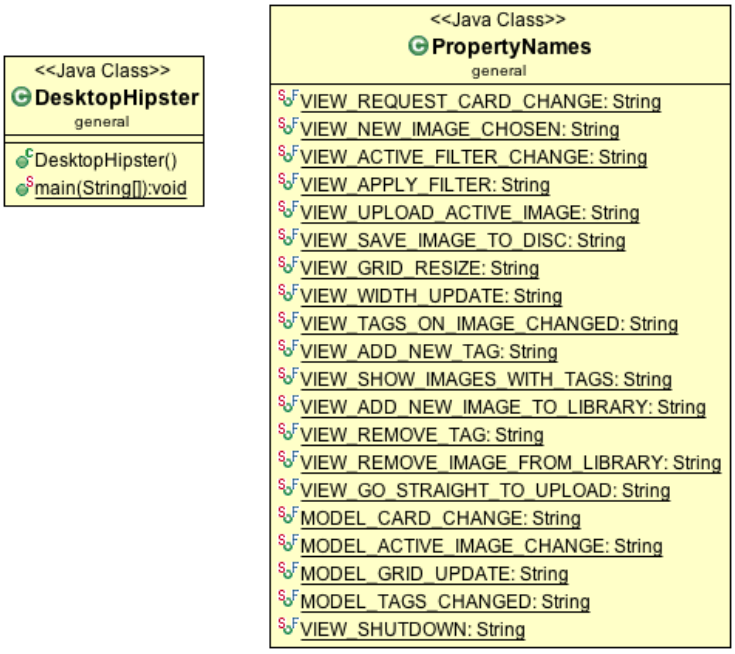


Figure 3: General package diagram

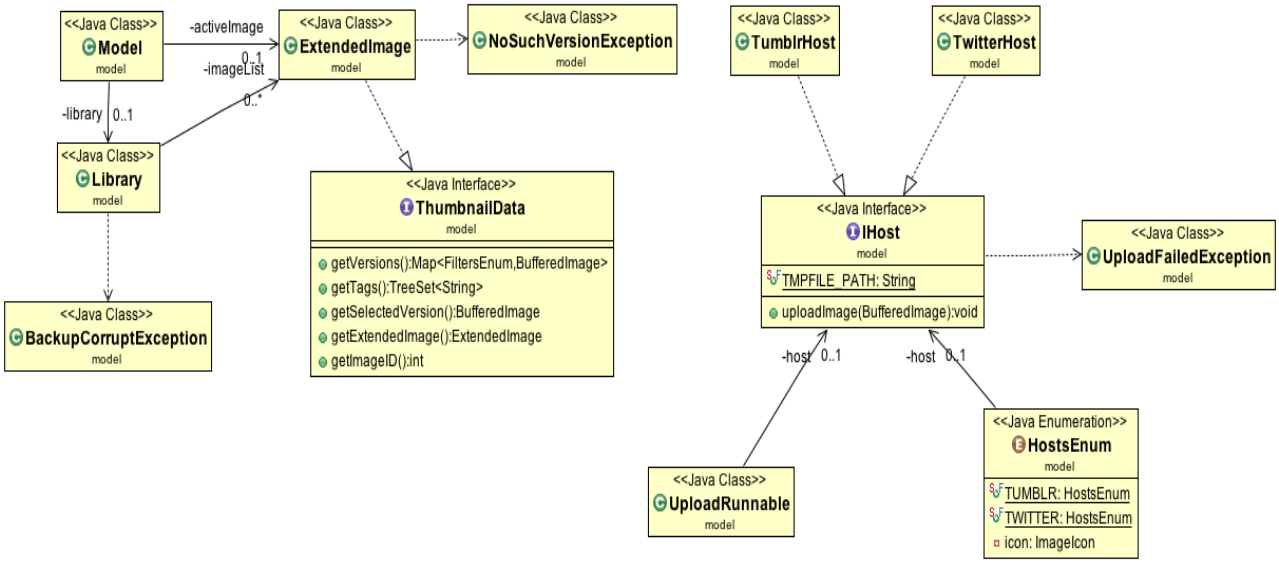


Figure 2: Model package diagram

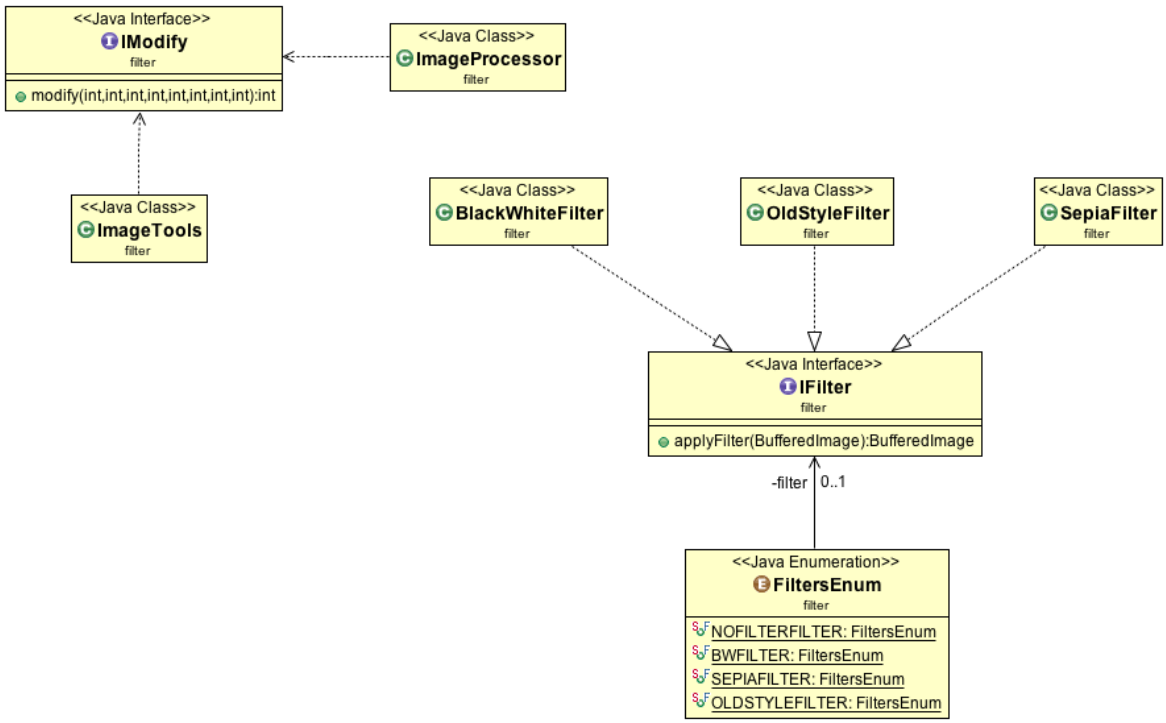


Figure 5: Filter package diagram

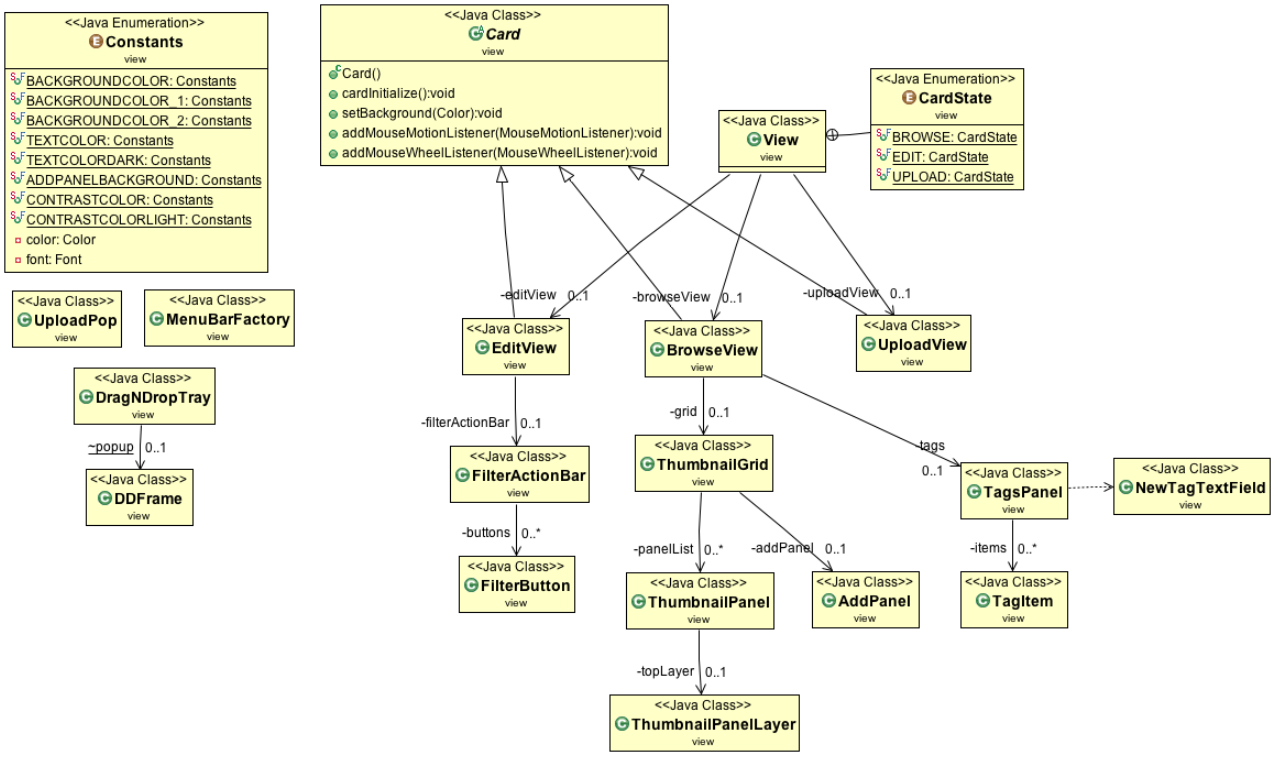


Figure 4: View package diagram