Software Development Methods [January 8, 2014] Fall 2013

Final

Note

This is an open-book exam. You may consult any sources (including online ones), but discussion with others is strictly forbidden.

Problems

1. Suppose you are developing a gallery app for mobile devices and desktop computers. The app supports image formats including JPG, PNG, TIFF, BMP, etc. You focus on the features and ease of use of the app, and reading/decoding of different image formats is supported by third-party libraries, each with its own API (application programming interface).

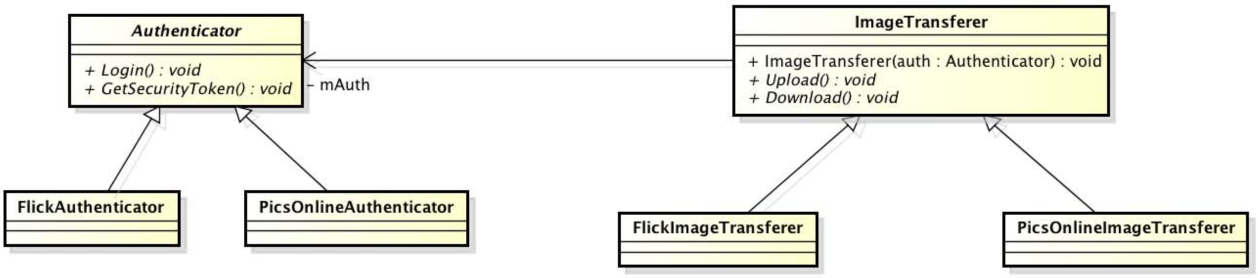
a) (4%) What design pattern will you use to increase the flexibility of your application so that third-party library calls do not appear throughout your code base?

b) (6%) Suppose you have the following two image decoder libraries and your abstract image decoder. Please apply the pattern to make use of the two image decoder libraries. Please also provide a class showing how third-party library calls are insulated from your code base.

```
// Forward declarations. Definitions in another header file.
class Size;
class Buffer;
class InputStream;

class ImageDecoder {
public:
  virtual Size* GetSize() = 0;
  virtual Buffer* Decode(InputStream* input) = 0;
  virtual ~ImageDecoder() {}
};

// Third-party library.
namespace vendor_impl {
class JPEGImageDecoder {
public:
  Size* GetImageSize(); // Implementation omitted.
  Buffer* DecodeImage(InputStream* input); // Implementation omitted.
};

class PNGDecoder {
public:
  Size* GetDimensions(); // Implementation omitted.
  Buffer* DecodeWithInputStream(InputStream* input); // Implementation
omitted.
};
}  // namespace vendor_impl
```

2. Now suppose in the next revision, you want to add image upload/download feature to your gallery app. The app supports several popular online photo album services, including “Flick”

and “PicsOnline”. Here are the classes that interact with the online services:



And here is the function that creates and uses the Authenticator and ImageUploader classes:

```
void UploadeImages(vector<InputStream*> images)
{
  Authenticator *auth = NULL;
  ImageTransferrer *transferrer = NULL;

  if (onlineService == kFlick) {
    auth = new FlickAuthenticator();
    transferrer = new FlickImageTransferrer(auth);
  } else if (onlineService == kPicsOnline) {
    auth = new PicsOnlineAuthenticator();
    transferrer = new PicsOnlineImageTransferrer(auth);
  } else {
    throw UnknownOnlineService;
  }
  // Use the authenticator and transferrer.
}
```

The function is not well-designed because the function body can grow longer and longer with more online services supported. It should not concern what the concrete Authenticator and ImageTransferrer classes are, but the names of the concrete classes litter in the function body.

a) (4%) What design pattern can you use to make the function more flexible so that the function does not instantiate and depend on the concrete classes?

b) (6%) Please provide your design in UML for creating these concrete classes.

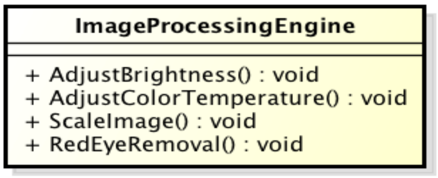
3. (10%) It is common that the user manages thousands of pictures with the gallery app. One photo album may contain hundreds of pictures. To let the user browse the pictures smoothly, the app generates, caches, and uses scaled-down images in the thumbnail view. Suppose you have the following ill-designed Image class. What design pattern can be used to move the thumbnail-related logic out of the Image class and control the access to the full-size image in the thumbnail view? Please provide a class diagram for the refactored classes.

```
class Image {
public:
  void Display() {
    if (mCurrentDisplayMode == kThumbnail) {
      if (! HasCachedThumbnail()) {
        GenerateThumbnail();
      }
      LoadThumbnail();
    } else if (mCurrentDisplayMode == kFullSize){
      LoadImage();
    } else {
      throw UnknownError();
    }
    ShowLoadedImage();
  }

private:
  bool HasCachedThumbnail();
  void GenerateThumbnail();
  void LoadThumbnail();
  void LoadImage();
  void ShowLoadedImage();

  enum DisplayMode {
    kThumbnail,
    kFullSize
  };

  DisplayMode mCurrentDisplayMode;
};
```

4. Now suppose that the gallery app is popular. Many users start to request image editing features so that after images are added to the gallery, the user may apply editing actions such as adjustments to brightness, white balance, red-eye removal, image cropping and rotation, etc.

The app supports user-defined action, which contains a series of basic editing actions. For example, a user-defined action named “Batch1” may contain consist of:
1. Brightness +2
2. Color Temperature +1200
3. Resize to 800x600

The image editing functionality is provided by a third-party library. It has a facade class, ImageProcessingEngine as follows:



a) (4%) What design pattern can be used to support the composition, and delayed execution of the actions to facilitate batch processing?

b) (6%) Please provide your design in UML for applying the pattern. Please use “AdjustBrightness”, “AdjustColorTemperature” and “ResizeImage” as concrete actions in the UML diagram.

5. (20%) The SQL Injection is one of the most common types of Web application security vulnerabilities/risks. What is the main cause of this security risk? How can it be prevented? Why is “prepared statement” an effective means to prevent such risks?

6. (10%) How is formal logic related to software verification?

7. Please provide a precise description, using logical formulae, for each of the following requirements. The functions/constants and predicates you may use are: 1, <, =, ≤, plus those introduced in the requirement statements. Make assumptions where you see necessary.

a) (10%) A[m] is the second largest element in array A[1..N].

b) (10%) Array B is a permutation of Array A (assuming both have N elements, indexed by 1 through N).

8. (10%) In the automata-based model checking, automata (representing systems/modules or properties) need to be composed either by asynchronous product or synchronous product. Why do we need two different kinds of composition? Please try to explain as comprehensively as possible.