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 the core system user interface of a mobile operating system. The system or some installed apps may become unstable when the system runs out of internal storage. Therefore, the system needs to notify the user when the mobile phone is about to run out of its storage.

The system has the following interface that allows you to get notified when the internal storage runs below some specified level:

```
class SystemStorageManager {
public:
   // Get notifications when free space is < aFreeSpace.
   void AddStoragePressureWatcher(StoragePressureWatcher* aWatcher, unsigned long
aFreeSpace);
   void RemoveStoragePressureWatcher(StoragePressureWatcher aWatcher);

private:
   // Will call StoragePressureWatcher::OnDiskPressure() in this method.
   BroadcastStoragePressure();
};

class StoragePressureWatcher {
   // Called when free space is < aFreeSpace.
   virtual void OnStoragePressure(unsigned long aFreeSpace) = 0;
};
```

a) (5%) What design pattern is used in the above classes? Please also name the roles (participants) the above classes take in the pattern.

Suppose class SystemStorageManager is expensive to instantiate. An instance of SystemStorageManager takes considerable memory and some other resource that are of limited amount in the whole system. To prevent unnecessary waste of system resources, you decide to restrict the instantiation of SystemStorageManager to the following interface:

```
namspace service {
   SystemStorageManager* GetSystemStorageManager();
}
```

b) (5%) What design pattern can be used to limit the number of SystemStorageManager instances so that we can reduce the used resources to a minimum?

c) (5%) Please provide your implementation of service::GetSystemStorageManager() for the requirement in b)

2. Suppose you are developing the camera app on the phone. The app supports installable filters that allow users to apply effects when taking photos or selfies. For example, the "RemoveFog" filter may enhance the color for a foggy scenic photo. The app also allows the user to apply multiple filters at the same time. The system provides the following interface for filter developers:

```
class ImageData {
public:
   // The methods to manipulate the data.

private:
   // The pixels of the image.
};

class ImageFilter {
public:
   virtual void Apply(ImageData* aImageData) = 0;
};

// The class to perform post-processing in the camera app.
class ImagePostProcessor {
public:
   void AddImageFilter(ImageFilter* aFilter);
   void RemoveImageFilter(ImageFilter* aFilter);
};
```

a) (5%) What design pattern is used in implementing the concrete filters so that the user can extend the functionality of the camera app?

Suppose you now have the following third-party library for image processing. The library provides the following interface:

```
class ColorEnhancer {
public:
   // Enhancement for a foggy photo.
   void RemoveFog(ImageData* aImageData);

   // Make dark part of the photo brighter.
   void Brighten(ImageData* aImageData, float amount);
};
```

b) (5%) Please apply the pattern and implement the "FogRemover" filter class.

Suppose that installable camera filters become very popular. You then decide to strengthen the security of the system to prevent malicious filters from doing harms to the user. This can be done via *sandboxing,* where the system runs these filters in a secure environment by:

(1) executing the filter code in a separate process, and
(2) forwarding the API calls in the filter process to the system to perform permission checks.

If the filter uses some API that it doesn't have permission to, the system rejects the access and aborts the filter (by killing the process). For example, if class BrightenerFilter, which should only access the image data, tries to get user information via:

```
// In BrightenerFilter::Apply(ImageData* aImageData):
ContactManger* manager = system::ContactManager::GetInstance();
UserContact* contact = manager->GetUserContactByName("John Smith");
```

The system kills the process when the filter calls system::ContactManager::GetUserContactByName() (for simplicity, we don't check against system::ContactManager::GetInstance()).

To implement this new security mechanism without requiring the existing filters to change, you need to keep the interface of ContactManager unchanged and forward system::ContactManager::GetUserContactByName() to the system **transparently** using the proxy pattern.

c) (5%) Suppose you have the following interface for sending a request to the system, please apply the pattern and provide your design in class diagram for sandboxing class system::ContactManager on the filter side.

```
class RemoteContactManager {
public:
   // Send the request to the system to get user information.
   // Returns nullptr if the specified user is not found or multiple users are found.
   UserContact* SendGetUserContact(string aUserName);
};
```

3. (10%) Draw a UML sequence diagram to describe the activities of a typical login procedure that involves a single sign-on authentication service.

4. (20%) Construct an abstract data model for a recommendation letters management system for university admission applications that has to meet the following requirements:

- ✓ An applicant may be applying for several different programs/universities, each of which usually requires two or three recommendation letters.
- ✓ A recommendation letter may be in different states, including "not started" (or "requested", resulted from a request newly made by the applicant), "incomplete", "complete", and "submitted".
- ✓ A submitted letter can be viewed, but not edited, by the recommender.

- ✓ A recommender may write a recommendation letter for several different applicants.
- ✓ Contact information is maintained for programs/universities, recommenders, and applicants.
- ✓ A recommender must have an affiliation and a job title.

Please use the UML as much as possible when describing the model. State the assumptions, if any, you make for your design.

5. (10%) Why is logic relevant to (more rigorous) software development?

6. Please provide a precise description, using logical formulae, for each of the following requirements. The functions/constants and predicates you may use are: 0, 1, $<$, $=$, $\leq$, plus those introduced in the requirement statements. Make assumptions where you see necessary.

a) (5%) The elements from A[0] through A[i] in array A[0..N-1] (of integers, indexed by 0 through N-1) are sorted in ascending order, and so are elements from A[i+1] through A[N-1].

b) (5%) Array B is a result from merging (as in the merge sort algorithm) the preceding two sorted partitions of Array A.

7. Answer the following questions regarding software testing.

a) (10%) What is black-box testing and what is white-box testing? Explain the terms and, for each, give an example in the context of programming assignments/projects of a course.

b) (10%) What is equivalence partitioning? Why is the notion important?