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 designing the order processing subsystem of a concert ticket selling system. An order is identified with the timestamp when the transaction is completed. You proposed the following design for querying an order using the timestamp identifier:

```
class Ticket {
   // Details of the ticket, including date, time, seat, etc.
};

class Order {
public:
   std::Vector<Ticket> GetTickets();
};

class OrderProcessor {
public:
   // Timestamp is the number of seconds since the epoch time (1970/1/1 0:0:0)
   std::vector<Ticket> GetOrder(Timestamp timestamp);
};
```

When you performed functional tests, the design worked fine. Then when you performed load tests to put the system under high traffic loads with 100 concurrent users making orders continuously, the method OrderProcessor::GetOrder() didn't always behave correctly. After reviewing the design, you realized that this design is faulty.

a) (4%) What is the problem with the above design? Why is the bug only seen when the system is tested with high loads?

b) (6%) What design pattern can be used to solve the design fault? Describe how the pattern solves the problem.

2. To better use the multiple cores that current CPUs have, we often create multiple software threads in a process, where threads may execute on different physical CPU cores at the same time. Thread pooling is a technique used to simplify multithreading. A thread pool class typically has a method AddTask() for its client class to add work to be done by the thread pool:

```
// The interface that represents work to be done by the thread pool.
class Task {
   virtual void Run() = 0;
   virtual ~Task() {}
};

class ThreadPool {
   // The interface for any client class to add work wrapped
   // in a concrete Task class to be done by the thread pool.
   void AddTask(Task* task) {
      // AddTask() adds |task| to a queue that is shared by
      // all threads the ThreadPool manages.
      // It notifies an idle thread that is waiting for a task to run.
   }
};
```

A client class typically uses the thread pool as follows:

```
class ConcreteTask : public Task {
   void Run() override {
      // Work to run on the thread pool.
   }
};

// Get a thread pool to use.
ThreadPool* threadPool = GetThreadPool();
ConcreteTask* aTask = new ConcreteTask();
threadPool->AddTask(aTask);
```

a) (4%) What design pattern is used to let the client class create and send the work to be done by another class (ThreadPool).

b) (6%) Sometimes some related tasks cannot be run concurrently and need to be run sequentially. Suppose we want to use the thread pool to run a series of tasks sequentially on a thread of the thread pool. Please implement a utility class, SequentialTasks, to support the following use case:

```
SequentialTasks* sequentialTasks = new SequentialTasks();
// |task1| and |task2| are of type Task*.
sequentialTasks.Add(task1);
sequentialTasks.Add(task2);

// Run |task1| and |task2| sequentially on the thread pool.
threadPool->AddTask(sequentialTask);
```

3. Suppose you are working on an instant messaging app “SnapTalk”. The app supports “shared album” for multi-user chat groups. For a large chat group, the shared album can easily contain hundreds of photos. Browsing through the album is slow because 10-megapixel cameras are a commodity for smartphones nowadays. Pictures taken using these cameras are large and slow to load and display.
You observe that only a few users zoom in the photos to a 1:1 scale. Most people just skim

through low-resolution thumbnails. This makes it possible to enhance the user experience by showing thumbnails at first and loading full-sized pictures when the user wants the app to.

Suppose the following interface is used to display a picture in your app:

```
class AbstractImage {
public:
   // Returns the width of the image.
   virtual int GetWidth() = 0;
   // Returns the height of the image.
   virtual int GetHeight() = 0;
   // Loads the image.
   virtual void Load() = 0;
   // Zooms the image to |zoomFactor|.
   virtual void ZoomTo(float zoomFactor) = 0;
};

class Image : public AbstractImage {
public:
   // Returns the full width of the image.
   int GetWidth() override;
   // Returns the full height of the image.
   int GetHeight() override;
   // Load the Image from local storage or from the server.
   void Load() override;
   // Zoom the image.
   void ZoomTo(float zoomFactor) override;
};
```

The AbstractImage interface is used in many places in your app. You want to implement the thumbnail class in a transparent way to the client classes. The thumbnail class can be used anywhere the AbstractImage is used. The thumbnail class uses the local cache initially and controls when we request the full image (use a zoom factor that is larger or equal to 1)

a) (4%) What design pattern can be used to implement the thumbnail class?

b) (6%) Please provide the class diagram showing the relationship between class AbstractImage, Image, and Thumbnail.

4. Answer the following questions regarding software verification and validation.

a) (4%) What is the main difference between verification and validation?

b) (6%) Why do we need both of them in an adequate software development process? Please explain the reasons using examples.

5. (20%) Suppose your team is contracted to develop a reservation system for a car rental company. To get a good start, you first try to construct an abstract data model for the system.

Below are the requirements you have gathered from interviews with your client:

- ✓ The company has a few hundred vehicles.
- ✓ Each vehicle is classified into several types: sedan, sports car, van, truck, etc., where a sedan is further categorized as subcompact, compact, mid-size, and full-size.
- ✓ A car is uniquely identified by its plate number, but may be parked in any available parking lot that is also uniquely numbered.
- ✓ Cars are rented by hours, but charged by days (less than 24 hours is counted as a day).
- ✓ The daily rate of a car is priced according to its type and is usually higher during a weekend.
- ✓ The company is very busy, so they want to be able to check quickly the availability of a particular car for particular dates within three months.
- ✓ Every reservation of a car, if for a multiple-day stay, must be of contiguous days.
- ✓ A customer must have a full name and an email address.
- ✓ A reservation can be of four states: incomplete, complete, checked-out, returned.

Please use the UML as much as possible when describing the model. There MUST NOT be any many-to-many relation in the model. State the assumptions, if any, you make for your design.

6. (10%) What is the essence of the weaknesses in a program that an SQL (or command, in general) injection attack exploits? How does that essence explain why prepared statements are effective against such injection attacks?

7. (10%) Why should you, for security reasons, simply discard an illegal input rather than try to modify it so that it complies with the input requirements? Please elaborate with examples.

8. Answer the following questions regarding software testing.

a) (5%) Why can a black-box testing never be static? Why is a white-box testing not necessarily static?

b) (5%) What is equivalence partitioning? Please explain with an example. Why is the notion important?

9. 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, <, =, ≤, plus those introduced in the requirement statements. Make assumptions where you see necessary.

a) (5%) The array A[0..N-1] represents a max heap with A[0] as the root.

b) (5%) The array A[0..N-1] is cyclically sorted in an increasing order.