Final

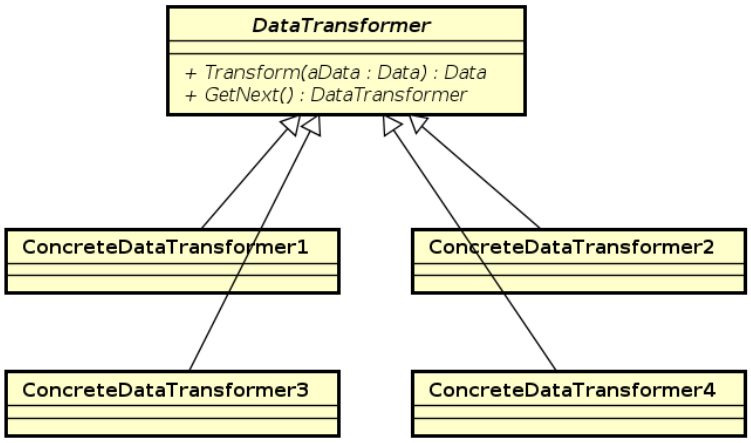
Note

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 data transformation application. The application can be used to extract data from unstructured text input, apply one or more transformations to the extracted data, and then output the result in one or more supported formats.

Suppose the application supports 4 concrete data transformations, represented as 4 concrete DataTransformer classes:

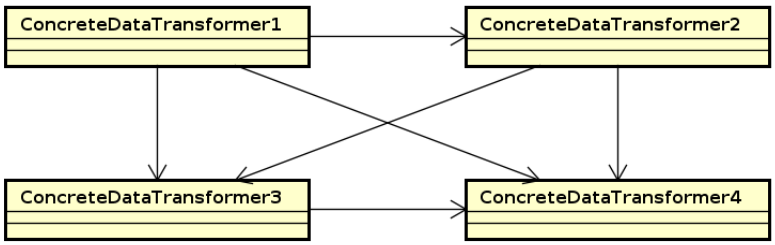


Each concrete DataTransformer may decide what transformation to apply after its Transform() is finished. For example, ConcreteDataTransformation1 may decide the next transformation to be ConcreteDataTransformation 2, 3 or 4 depending on the result of its transformation:

```
class ConcreteDataTransformer1 : public DataTransformer {
public:
  virtual Data* Transform(Data* aInput)
  {
    // Step 1. Apply transformation to aInput.
    Data* output = Process(aInput);

    // Step 2. Decide next transformer by the output of this
transformation.
    if (output->Contains("Data for transform 2")) {
      if (output->HasSomeAttribute()) {
        mConcreteDataTransformer2->EnableOptionalSubstep();
      }
      mNext = mConcreteDataTransformer2;
```

```
    } else if (output->Contains("Data for transform 3")) {
      mNext = mConcreteDataTransformer3;
    } else if (output->Contains("Data for transform 4")) {
      mNext = mConcreteDataTransformer4;
    } else {
      mNext = NULL;
    }
    return output;
  }
  virtual DataTransformer* GetNext()
  { return mNext; }

private:
  DataTransformer* mNext;

  ConcreteDataTransformer2* mConcreteDataTransformer2;
  ConcreteDataTransformer3* mConcreteDataTransformer3;
  ConcreteDataTransformer4* mConcreteDataTransformer4;
};
```

The dependencies between the concrete DataTransformer classes are shown in the following class diagram:



a) (4%) Why is this design inflexible when we want to support more kinds of data transformation by adding more concrete DataTrasformation classes?

b) (2%) What design pattern can be used to solve the flexibility problem in a)?

c) (4%) Please provide your new design in a class diagram.

2. Many operating systems provide the sockets application programming interface (API) for development of network clients or servers. The API provides fine-grained control over network connections. For example, to make a client connection, you need to do the following:

```
// First, create a integer socket file descriptor with the socket()
call:
int socketFileDes = socket(AF_INET, SOCK_STREAM, 0);

// Then you fill in the fields struct sockaddr_in for the server
that you
// want to connect to:
struct sockaddr_in server;
server.sin_addr.s_addr = inet_addr("192.168.0.10"); // Fill in the
IP address.
server.sin_family = AF_INET; // AF_INET for Internet connection.
```

```
server.sin_port = htons(80); // Fill in the port number.

// And finally you connect to the server by making the connect()
call:
connect(socketFileDes, (struct sockaddr *)&server , sizeof(server));
// Phew! You are finally connected to the server. You may
send/receive data
// using the socket file descriptor now.
```

a) (6%) Suppose you want to hide the gory details of using this **low-level** socket API in a simpler interface, what design pattern can be applied for this goal?

b) (4%) Please apply the pattern so that making an Internet connection to port 80 of host 192.168.0.10 can be accomplished by the following:

```
ClientConnection* connection = new ClientConnection();
connection->Connect("192.168.0.10", 80);
```

3. Suppose you have a class named MessageArchiver, which has a method named Archive for saving text messages to disk:

```
bool MessageArchiver::Archive(Date* aStartDate, Date* aEndDate) {
  std::vector<Message>* messages = GetMessagesByDate(aStartDate,
aEndDate);
  if (messages->empty()) {
    return true;
  }
  CompressedData *compressedData = Compress(messages);
  if(!SaveToLocalDisk(compressedData)) {
    return false;
  }
  delete compressedData;
  delete messages;
  return true;
}
```

Later you decide to add another method, MessageArchiver::ArchiveToServer() to save the compressed messages to a predetermined server. MessageArchiver::ArchiveToServer() differs from MessageArchiver::Archive() only in one line of code:

```
if (!SaveToLocalDisk(compressedData)) {
```

which needs to be replaced by

```
if (!SendToServer(compressedData, GetServerAddress()) {
```

So you copy and paste the implementation of MessageArchiver::Archive() and modify the line. Mission accomplished.

a) (4%) Do you think this is a good way to implement MessageArchiver::ArchiveToServer()? Why? What pattern can be used if you want to improve the design?

b) (6%) Please provide your implementation in C++.

4. (10%) Draw a UML sequence diagram to describe the activities of a typical login procedure that involves a single sign-on authentication service.

5. (20%) Suppose your team is contracted to develop a booking system for a coworking space. 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 coworking space has a few hundred seats, each of which may be rented by days (from one day to a few months).
- ✓ Several seats may be grouped together and rented simultaneously for the same period of time.
- ✓ The coworking space also has a few meeting rooms that may be used by its members.
- ✓ An administrative staff as well as a person who has rented a seat is considered a member of the coworking space.
- ✓ Meeting rooms may be reserved only by hours.

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

6. Answer the following questions regarding software testing.

a) (6%) 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) (4%) What is test-to-pass and what is test-to-fail? Explain the terms and give an example for each.

7. (10%) What is the essence of the weaknesses in a program that an SQL (or command, in general) injection attack exploits? Why prepared statements are effective against such attacks?

8. 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) (10%) An array A of N integers (indexed from 1 through N) is a max heap. (Please use A[i], as usual, to refer to the i-th element in A.)

b) (10%) An array A of N integers is cyclically sorted in ascending order. (2, 3, 4, 0, 1 is a cyclically sorted list of integers.)