CSC407/2103 - Assignment 2
(based on a real problem)
Due March 26 in tutorial

You are a software architect in a large electronics company that develops a variety of scanners (MRI, CT, US (Ultrasound), CR (Computed Radiography), etc.). Your company is developing a new image review workstation called ScanView, which views scans, i.e. data sets acquired by a scanner. The following is a requirements specification for a subsystem of ScanView. Your deliverables are a design document that specifies a subsystem for converting incoming images to the required format and Java code that implements the design. Details of what to submit can be found at the end of the Requirements.

REQUIREMENTS
DataConverter Sub-system

Overview

The deliverable for this project will be a DataConverter sub-system that will integrate with our ScanView workstation, currently in design. ScanView is a UNIX workstation that can be installed at various locations throughout the hospital environment, allowing clinicians (radiologists, surgeons, Intensive Care Unit doctors, etc.) to have local access to scans, wherever the scans were done. Some of the scans are done on our firm's equipment, but in many cases they were done on other vendors' equipment within the hospital, or externally, at another facility. ScanView will be connected to all of these systems via a TCP/IP based LAN or WAN. Other ScanView subsystems will communicate with all of these systems and manage the user interface. The DataConverter subsystem is part of the NetworkManager process, which manages all the data streams associated with ScanView's connections to the world. The source data itself may be stored on-line in a variety of databases, or on long term storage media (e.g. optical disks). It will be retrieved by remote systems and sent to ScanView. The responsibilities of DataConverter (details below) are to process images as they arrive and convert them to the internal data format required by ScanView, and store them in the local system's database.
You can assume:

- A scan is composed of multiple images, each representing a 2-dimensional cross sectional view of a region of a person's anatomy. The orientation and location of the cross sectional view can be arbitrary, with the constraint that all images in a single scan will have the same orientation, and a different location.
- All images in a single scan have the same data format.
- All data formats have this much in common: they have a header, consisting of descriptive data, and the image data itself.

R1: Asynchronous behaviour:

R1.1 The images will arrive one at a time. DataConverter must convert the images as they are provided by NetworkManager and make them available to the application via the database.
R1.2 Serialization: NetworkManager will ensure that the scans are provided to DataConverter one at a time. If an image from a new scan is provided prior to the previous scan being complete, DataConverter should take this as an error condition, stop the previous scan's conversion, clean up and create an event (per R4)

R2: Output data:

R2.1 data storage: As each image is successfully decoded DataConverter will use a scanStore object to store the image in the local database. scanStore provides the following interface for this purpose:
ScanStore(int xDim, int yDim, int numImages) : the constructor for a scanStore object takes the x and y dimensions of the pixel array, and the number of images in the scan, to allow it to initialize the private data structures it will use for this purpose.
scanStore.setXyz(int imageId, dataElementXyzType dataElement) : there will be one such method for each entry in the data dictionary [see the Data Dictionary below]. The imageId references a particular image in the scan (0< imageId <= #images in the scan).
scanStore.commit( ) : once all data elements have been stored for all images in a scan, the commit operation should be called to signify the completion of the transaction.
scanStore.rollback( ) : if DataConverter detects a case where a partially stored scan should not be stored, the rollback method must be used. This will ensure that the database disposes of any disk resources that may have been allocated and is left in a consistent state.

R2.2 cancellation: DataConverter must allow the data conversion to be cancelled. The granularity can be at the image level – in other words, it does not have to cancel in the middle of decoding an image, but if a cancel event has been received, it should not proceed to the next image, but rather cease work on the current scan and clean up.

R3: Data formats:

R3.1 Accept or reject: DataConverter must provide an interface that will accept information from NetworkManager describing the incoming set of images that are to be processed (the information will typically be scanner make and model and software version number, and the number of images). This interface must include a mechanism for DataConverter to advise the caller to reject the data, if it does not have the ability to decode it.
R3.2 Ease of extension: The design should accommodate easy extensions for data formats not currently supported. In particular, a file should list the names of Java classes that will perform the data conversions, to be dynamically loaded at runtime. An object instantiated from the class will describe the image type supported and provide the conversion code.

R4: Notification: DataConverter shall provide a means by which any number of application level entities can be notified of the following events, should they choose:

- Scan conversion started
- Progress (an estimate of the % complete)
- Scan conversion complete
- Completion status (success or error code)
- Abnormal condition (network not responding)

The NetworkManager handles all IPC that may be required, so that DataConverter can assume all notifications are within the same process.

R5: Data Consistency: Because these are medical images and we need to take extraordinary steps to ensure the integrity of the data, before DataStore.commit( ) is called, DataConverter will do the following consistency check to ensure that the data is correct: ensure that the z-locations of the images (refer to the Data Dictionary) form an ascending or descending sequence. If not, reject the scan, and create an event (per R4) with relevant details.

---End of Requirements-----

Additional Information:

Image data dictionary (for output):
An Image will consist of the following attributes (note that this represents a considerable simplification of what would normally be stored as an image[1])

Type: string	the modality. A string from the following set: "MR", "CT", "US", "CR"
Num: integer	Image number (in scan)
Length: integer	Number of images in scan
xSize: integer	the number of pixels per row in the image
ySsize: integer	The number of rows in the image
xNorm	3 floating point numbers representing the unit vector that is normal to the plane defining the orientation of the images.
YNorm	
zNorm	
Zloc: float	The z-coordinate of the image (relative to the scanner)
Patient: string	The name of the patient
ScanDate: Date	The time& date of the scan
ImageData: array	A 2d array of pixels (dimensions are Xsize by Ysize), each pixel is16 bits deep.

What you know because of your domain knowledge:

- The DataConverter subsystem will define the interface that ScanView developers will code against. This interface must be nailed down and solid early in the development cycle, as changes to it will affect ScanView's deliverables. Therefore it is important that the DataConverter interface shield ScanView from changes as far as possible.
- See the attached diagram for the DataConverter's role in the System Architecture
- From earlier projects you know that one of the problems you have to deal with (which will drive your design decisions) is the large number and variety of image formats and sources the images come from. You have dozens of Java classes that can be used or adapted to decode (i.e. read and parse) image files from a large number of sources. One of the aims of this design is to re-use – not re-write - this code. [Note – for the assignment, you will be implementing any of these classes you require – see Testing below].

For this assignment: First perform an OOA of the domain using information obtained from the Requirements and your domain knowledge. Submit UML Class diagrams, Use Case diagrams, Sequence Diagrams and associated explanations. Use this as the basis of your design. The design should make use of as many Design Patterns as you can usefully apply. Provide UML Class, Object and State Diagrams, with accompanying descriptive text. In addition to the design, there will be an implementation step: in Java, code the classes you define, and provide a test harness. The database calls should be implemented as simple file operations; the Commit and Rollback operations can be stubs.

Testing: The aim of the assignment is not to spend a lot of time decoding a large number of image formats, but rather to come up with a flexible design that supports this task. For testing purposes, test with at least 3 different valid input formats that you will create. For simplicity's sake:

- One case can be a "pass-through" – i.e. already has the data elements in the required formats – you will still need an object that implements this.
- You can test with scans that have at most 3 images per "scan".
- You can construct the input formats simply by rearranging the data dictionary above (so "decoding" should be straightforward – simply map the input header elements to the correct internal data elements).
- For some cases, you can assume the ImageData is already in the correct format but for at least one test case, many MR and CT scanners produce 12 bits of data per pixel, where the pixels are packed 2

[1] In a real system, the number of attributes in the image, and their representations would be much more complex. In particular, there are a number of geometric descriptors related to patient position and orientation that you would likely need to convert from the native scheme to the scheme used by ScanView.

pixels per 3 bytes. Construct at least one set of test images with the pixel data in this format, so that the DataConverter will need to unpack the pixels and store them in the internal format.

Also, test for as many exceptions as you can.

- Ensure that you have at least 1 test case where data consistency fails, to show that the code you implemented for this case works (R5).

In terms of importance:

- The OOA will be worth 20% of the mark.
- Focus on the Design (50% of the mark). It should be simple, cover all the requirements, and make effective use of Design Patterns (you should be able to apply at least 4, but this is not cast in stone – perhaps you can find an elegant solution that makes use of fewer). Design Patterns will be marked according to correctness, appropriate use, and quality of the associated descriptions. Diagrams should be clear, well organized, and documented. Anticipating problems with the design or requirements as stated, and proposing improvements or alternatives may make you eligible for bonus marks.
- Next, focus on the implementation (but not at the expense of the Design). (20% of the mark)
- After you feel you have a robust design and code that compiles, worry about testing. (10% of the mark).