Assignment 1 – An object-oriented library for sparse 2-dimensional matrices

Many engineering problems involve matrix manipulation; in fact, we might argue that engineers are *domain experts* in the practical use of matrix manipulation techniques. Many engineering problems involve the use of large matrices often with high numbers of dimensions; but often these matrices are sparsely populated. This sparse characteristic allows us to utilize **more efficient and effectively techniques and representations**, which have been specifically developed for problems utilizing sparse matrix representations.

Unfortunately, Ruby has no standard library provision for sparse matrix problems - hence Assignment 1.

Your task is to….

1) Increase your knowledge in this domain
2) Based upon this increased knowledge, consider the problem from a number of design perspectives. And hence, answer the posed design questions.
3) Produce a detailed design (**by specifying contracts**) of the library (**using TestUnit or other Unit testing frameworks**) for implementation in Ruby
4) Construction the library or framework (in Ruby)

To assist you with this task, please ponder (**and answer**) the following questions as you go …..

- What is a sparse matrix and what features should it possess?
- What sources of information should you consult to derive features? Do analogies exist? If so with what?
- Who is likely to be the user of a sparse matrix package? What features are they likely to demand?
- What is a tri-diagonal matrix?
- What is the relationship between a tri-diagonal matrix and a generic sparse matrix?
- Are tri-diagonal matrices important? And should they impact your design? If so, how?
- What is a good data representation for a sparse matrix?
- Assume that you have a customer for your sparse matrix package. The customer states that their primary requirements as: for a N x N matrix with m non-zero entries
    - Storage should be ~O(km), where k << N and m is any arbitrary type defined in your design.
    - Adding the m+1 value into the matrix should have an execution time of ~O(p) where the execution time of all method calls in standard Ruby container classes is considered to have a unit value and p << m ideally p = 1

  In this scenario, what is a good data representation for a sparse matrix?
- Design Patterns are common tricks which normally enshrine good practice

    - Explain the design patterns: Delegate and Abstract Factory
    - Explain how you would approach implementing these two patterns in Ruby
    - Are these patterns applicable to this problem? Explain your answer! (HINT: The answer is yes)
- What implementation approach are you using (reuse class, modify class, inherit from class, compose with class, build new standalone class); justify your selection.
- Is iteration a good technique for sparse matrix manipulation? Is "custom" iteration required for this problem?
- What exceptions can occur during the processing of sparse matrices? And how should the system handle them?
- What information does the system require to create a sparse matrix object? Remember you are building for a set of unknown customers – what will they want?
- What are the important quality characteristics of a sparse matrix package? Reusability? Efficiency? Efficiency of what?
- How do we generalize 2-D matrices to n-D matrices, where n > 2 – um, sounds like an extensible design?

This assignment is worth 20% of the total marks of the CMPE410 assignments which are of course worth 100%.

This assignment must be completed in **THREE** parts.

Part 1: Your design rationale and system research – i.e. the written answers to the above questions must be completed by: **Tuesday February 2nd @ 5:00 p.m.**

That is provide a design report that articulates, in detail, the decisions and selections you have made during your design process with regard to the above questions.

Please note: you are expected to back up this design report within the System Design and the solution!!!
If your design report says "we will do A!"; then your set of contracts need to describe ALL of the constraints that these decisions imply and your code must do "A".

If not, you need to hand-in an augmented design report in Part 3, explaining the rationale as to why you have changed the design decision.

Part 2: Your System Design (written as a set of assertions) must be completed by: **Tuesday February 2nd @ 5:00 p.m.**

This must include for each class: A full set of class invariants and a set of pre and post conditions for each method within the class.

And they must be compliant with the TestUnit package for Ruby!!!

Part 3: Your system must be completed by: **Tuesday February 9nd @ 5:00 p.m.** and must be ready for demonstration in the practical session.

In addition to the practical demonstration, you are required to hand-in:

1) A detailed rational for ***any augmented decisions*** with regard to the above design questions.
   Augmentations are not necessarily a bad thing – but they are a learning process! So list them!
2) A list of deviations in the contracts implemented form the contracts specified in Part 2.
   Deviations are not necessarily a bad thing – but they are a learning process! So list them!
3) A copy of the code
4) A description of any additional testing beyond that described by your contracts.
5) A list of known errors, faults, defects, missing functionality, etc. …telling us about your system's limitations will score better than letting us find them!

Please hand in all components by emailing them to {jimm, zuhori}@ualberta.ca and hence all sub-components by definition must be machine readable. In addition,

- The Subject Line should be in a specific format - Assignment 1: Group 1 [, Part 1...]
- Filetypes should be only of .doc or .pdf (for non-code solutions)
- Filenames for code should be in ruby format (all lower cases, with words concatenated with a dash)
- For each assignment with a code section, there must be a primary executable file that is the main entry for other codes. This has to have the format: assignment1_main.rb (where 1 corresponds to the appropriate number in the assignment sequence). This main file should also contain a comment section with a list of the group members if necessary and some description of the runtime requirement of the code (e.g. commandline activation format)