CS 246 **Software Design and Development**

Togetherness

Collections *Exploration*

Goal

The purpose of this exploration is for you to learn how to use collections of many different types and to collaboratively apply the **gersy principle** toward the specific end of putting together a useful document processing utility.

Requirements

Design and implement a java solution to the overarching problem of extracting and parsing the text from a Microsoft Word 2007 document (.docx file extension). More specifically, this includes retrieving the data from the network, uncompressing the compressed entries comprising the .docx data, finding the right contents and parsing the XML contained therein.

Once thus retrieved and "collected" somehow, you must display the content of the textual parts of the document as plain old text — i.e., with no formatting, and with each paragraph output as one long line, where one blank line separates paragraphs and one additional blank line separates parts – e.g., main text and footnotes.

The specific document to test is found at the URL

`http://emp.byui.edu/neffr/docs/UML for Java Programmers.docx`

This document is **not** to be saved as a file on your hard drive, but rather retrieved by use of the java "net" library to open a URL connection to it. Nothing you extract/parse out of the document should be saved to disk either — you must do it all in memory.

Build and test your code in the Linux Lab via the command

```
sh catdocx.sh
```

Among other things, this script will compare actual versus expected output. If there *is* a mismatch of actual versus expected output, you'll see something like the following:

```
=====================================================================
diffing actual output with expected output...
---------------------------------------------------------------------
1a2
>
28c29
< I don't know how to parse word/footnotes.xml
---
>  Note that this is the kind of thing that causes Uncle Bob to "rant"!
---------------------------------------------------------------------
```

The '<' in front of a line identifies actual output, whereas the '>' identifies expected output. As visual inspection shows, they are different.

Grading Criteria

The points breakdown below is meant to guide you to the optimal solution.

10	Correctness of solution. The output revealed by the supplied `catdocx.sh` script will confirm if the output requirements were correctly met.
10	Supported an optional "wrap" parameter governing where lines are to be wrapped (newline whitespace characters inserted), with a default value of zero indicating that no wrapping is to be done.
10	Correctly used the many `java` packages/libraries you must somehow figure out how to use to fulfill the other requirements.
10	Used the `umlet` tool to create UML class diagrams of your design. (Assert that you actually did a design before starting to code!)
10	Created a single (or multiple) `.uxf` file(s), together with the derived `.jpg` file(s) which are submitted with your code.
10	Code includes links to the `.jpg` versions of your UML class diagrams in the **javadocs** class comments.
10	Created UML class diagrams that are **professional-looking**, **concise** and **complete**. (Yes, it's possible to be simultaneously concise and complete!)
10	Diagrams show that your design is **elegant** (i.e., BOOPOPs- and BOODPAPs-compliant) with special consideration given to the *collections* involved.
10	Collaborated and applied the **gersy principle**. (Please identify your collaborators.)
10	Commented code and used approved style. (Pay attention to this!)
100	Total Points