CS1073 Assignment #7 - Winter 2024

Due: Thursday, April 11th by 12:00 NOON in the Assignment 7 submission folder in Desire2Learn. (See submission instructions below).

The purpose of this assignment is to:

- Give you some practice with two-dimensional arrays.
- Give you some practice with partially filled arrays of objects.
- Review some of the material from earlier in the course.

This assignment is to be done individually. What you hand in must be your own work. Incidents of plagiarism will be reported.

If you have questions about the assignment, you should first go to a scheduled help session. (Locations and times for all help sessions can be found on D2L). If you have attended a help session and the issue is unresolved, you may contact your course instructor. You are NOT to discuss this assignment with anyone else (including your classmates).

---

1. Code Decryption[1]

Note the following requirement: You must use a two-dimensional array in your solution.

You are working as a codebreaker for an intelligence agency. Your informers have uncovered a new method used by some adversaries (and allies) to transmit encrypted messages. You've been tasked with writing an application that performs the decryption and outputs the unscrambled messages. The encryption is performed as follows:

The number of columns is selected first. Afterwards, the message (letters only) is written across the rows alternating left-to-right and right-to-left, and padding with extra random letters so as to make a rectangular array of letters.

[1] Based on the 2004/2005 ACM Regional Programming Competition in the "North America – East Central" region

For example, if the message is “Find the secret de-coder ring!” and there are five columns, the following would be written down:

```
f i n d t
c e s e h
r e t d e
r e d o c
r i n g x
```

Note that only letters are included and written all in lower case. In this example, the character `'x'` was used to pad the message to make a rectangle; however, any other letter could be used. Afterwards, the message is sent by writing the letters in each column, alternating bottom-to-top and top-to-bottom. So the above would be encrypted as:

```
rrrcfieeeindtsndedogxceht
```

Your task is to **recover the original message** (along with any extra padding letters) from the encrypted one.

Input

There will be multiple input sets. Input for each set will consist of two lines. The first line will contain an integer indicating the number of columns used. The next line is a string of lower-case letters. The last input set is followed by a line containing a single 0, indicating end of input.

Output

Each input set should generate one line of output, giving the original plaintext message, with no spaces.

Sample Input

```
5
rrrcfieeeindtsndedogxceht
3
kcnanaehrduowahtrelilngaaxyttnipsivofct
0
```

Sample Output

```
findthesecretdecoderringx
watchoutfordrevilheisplanninganattackxy
```

Testing

Once you have tested your solution with input that you type in by hand, please use the input file that we have prepared, named **`cypherText.in`** (available in Desire2Learn).

Recall: To feed in the input data from a file, simply use the **<** symbol followed by the name of the file that contains the input, e.g.:

```
java Decryptor < cypherText.in
```

Save the output that is produced in a text file named **plainText.out** when you run your program with **cypherText.in** as the input. Copy and past the output from the file into your report and include the output file (plainText.out) in your archive.

Recall: To feed the output data from running your code, simply use the **>** symbol followed by the name of the file that you want to contain the output, e.g.:

```
java Decryptor > plainText.out
```

Note: you can use both input < and output > on the command line to perform both input and output operations.

2. Spell Caster Classes

You are working on part of an application that keeps track of characters in a role playing game (RPG). You are working on the part of the application that keeps track of characters that are spell casters. You begin by writing three classes, named **Spell**, **SpellCaster**, and **SpellCasterApprentice**.

The **SpellCaster** class represents a spell caster that is allowed to hold spells in their spell book. You must record the name of each spell caster (e.g. “Latona”), along with their level (e.g. 5). Each spell caster is automatically assigned a spell caster guild membership number when they are entered into the system. These membership numbers are assigned in ascending (increasing) order, starting at the number 6000. Once assigned, each spell caster will retain the same membership number (it will not change).

For each spell caster, we need to keep track of what spells they currently have in their spell book. A spell casters start with no spells in their spell book, but may gain spells throughout the game. Their spell book can only hold up to a set number of spells. When a spell caster casts a spell from their spell book it is removed from their spell book. They may add new spells to their spell book as long as they have room. A **Spell** class represents the spells a spell caster can have in their spell book. You must record the name of the spell (e.g. “Invisibility”), the level of spell (e.g. 8) and if the spell requires material components or not (note: spell caster apprentices (which will be discussed in more detail later on) cannot add spells to their spell book if they require material components). Once these values are initialized, they cannot be changed. Three accessor methods must be included in the Spell class to retrieve the information when needed.

Mutator metohds must be provided in the **SpellCaster** class to allow the spell caster to add spells to their spell book and cast spells which will remove the spell

from their spell book. The level of the spell caster is used to determine what level of spells they can hold in their spell book. A spell caster can hold at most **7** spells in their spell book. A spell caster can only add spells to their spell book that are at or below their own level (e.g. a spell caster of level 5 can only have spells that are level 5 or below. The methods for adding spells and casting (removing) spells both return a boolean value to indicate whether the add or cast (remove) was successful. When a spell is cast, the array of spells representing the spell book will be adjusted so the spells are always stored in contiguous elements at the beginning of the array. It is not necessary to maintain the spells in any particular order.

The methods for adding and casting spells both receive a **Spell** object as their only parameter. When casting a spell, search through the array to find the spell provided, and if found, remove it from the array. Two **Spell** objects are considered to be equal if their name, level, and material component status are all the same.

Accessor methods should be provided in the **SpellCaster** class to retrieve the caster's name, level, and membership number. An accessor method should also be provided to retrieve a ***copy*** of the list of spells that a spell caster currently has in their spell book. This list is to be returned as an array, and this array will always be full. Since the list of spells in their spell book for a given **SpellCaster** can vary, the length of the array returned by this accessor method will also vary.

Along with spell casters, the designers of the application would also like to be able to include a more specific type of spell caster; these are spell caster apprentices (those who are learning to be spell casters). Apprentices have all the same information as spell casters (name, level, membership number, spell book). However, for each **SpellCasterApprentice** we have to record their supervisor which is a **SpellCaster**, and be able to access information about their supervisor when required. Furthermore, spell caster apprentices are not allowed to have spells in their spell book that require material components as they are not fully trained to use these types of spells yet.

The spells in a spell caster's spell book (including spell caster apprentices) may change over time. However, all other spell caster information, once set, should remain fixed.

Write the complete **Spell**, **SpellCaster**, and **SpellCasterApprentice** classes. For full marks, you must use inheritance and include complete Javadoc comments.

Also note: Data structures other than arrays (e.g. Vectors, ArrayLists, Linked Lists, etc.) are **not** permitted in this assignment (as they are not covered until CS1083).

Test your classes by using the **SpellCasterTestDriver** class that is provided. ***Do not change anything in this class***. For full marks you must make your classes compatible with this class, and all test cases must complete successfully.

For this assignment, only an electronic submission is required.

Your electronic submission (submitted via Desire2Learn) will consist of two files:

i. a written report. This should begin with a title page; your title page should include: the course (CS 1073), your section (FR01B, FR02B, FR03B, FR04B, FR05B or FR06B), the assignment number, your full name, and your UNB student number. That should be followed by the sections below, with each part clearly identified with a section heading. Include:

   i. the source code for the decryption problem (question 1),

   ii. the output produced when testing your decryption solution with `cypherText.in`,

   iii. the source code for the three requested classes for question 2, and

   iv. the output produced when testing your question 2 solution with the `SpellCasterTestDriver` class that was provided.

   This written report should be prepared using a word processor. (Copy & paste your java source code & output into the report document and add appropriate headings.) The report document should then be stored as a SINGLE pdf file and submitted to the appropriate drop box on Desire2Learn. (This pdf will allow the marker to write comments directly on your work to give you better feedback.)

   Note: Please name this report as follows: **YourName_As7_Report.pdf**

ii. an archive file (.zip or .tar) that contains all of your work for this assignment. Make sure that your archive includes the source code (.java files - in case the marker wishes to compile & run your code) and output files. This archive should be submitted as a single file to the appropriate drop box on Desire2Learn.

   Note: Please name this archive file as follows:
   YourName_As7_Archive.zip or **YourName_As7_Archive.tar**