Assignment 1

CS 1027
Computer Science Fundamentals II

Due date: February 4 at 11:55 pm

1. Learning Outcomes

In this assignment, you will get practice with:

- Implementing classes and various methods
- Working with objects
- Creating and manipulating arrays
- Using loops and conditionals
- Programming according to specifications

2. Introduction

One of the most extraordinary and iconic resources in all of science is the periodic table of elements. This table, which is used extensively in chemistry, physics, biology, earth sciences, and other sciences, contains all naturally occurring and synthesized chemical elements. The elements are shown as boxes organized in rows, called periods, and columns, called groups.

Each element in the periodic table contains an atomic number (i.e. 17), name (i.e. Chlorine), chemical symbol (i.e. Cl), atomic mass (i.e. 35.45), phase (solid/liquid/gas), among many other properties. Elements are also classified as metals, non-metals, or metalloids and these types help determine how multiple elements bond to one another to form compounds, which are combinations of two or more elements joining together. Figure 1 shows an example of what an element box in a periodic table might look like.

Figure 1. A depiction of one element box in a periodic table.

The full periodic table is illustrated in Figure 2. It may be small and hard to read in this document so you may want to look online for a larger image of the periodic table such as the website from which the image in Figure 2 was obtained (https://pubchem.ncbi.nlm.nih.gov/periodic-table/).

Assignment 1

CS 1027
Computer Science Fundamentals II

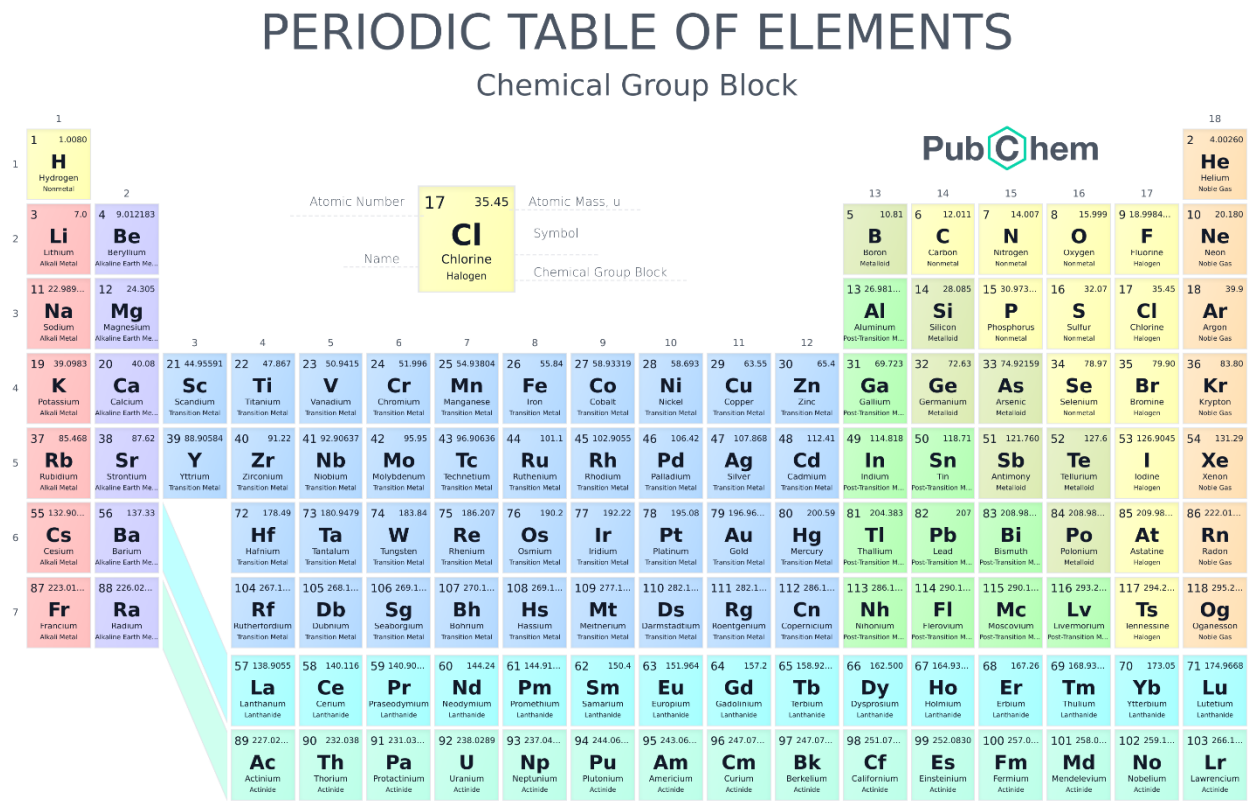

Figure 2. Periodic table image obtained from: https://pubchem.ncbi.nlm.nih.gov/periodic-table/

The two aqua-coloured rows at the bottom of the table are called the lanthanides and actinides, respectively. The lanthanides elements are numbered 57 to 71 and the actinides elements are numbered 89 to 103. They are special in that those whole rows have properties that fit with periods 6 and 7, respectively but they are normally shown as separate rows at the bottom as is the case in the above figure.

For this assignment, you will be creating a Java program to represent the periodic table and its elements and allows us to create simple compounds consisting of two elements that bond together. For this assignment, we will use 3 separate arrays to store the periodic table elements: one for the lanthanides, one for the actinides, and one for everything else. The last one will be a 2D (2 dimensional) array so that we have a grid-like structure to store the elements.

The spacing of the elements is important in these 3 arrays and must match the structure of the actual periodic table. For example, the top row (period 1) contains 2 elements: H (Hydrogen) and He (Helium) as you can see in the periodic table shown above. In the 2D array for your program, the top row of the array will contain the Hydrogen Element object at index 0 and the Helium Element object at index 17. All the array cells in between will be null. To clarify this point,

the 2D array will have capacities of 7x18 (7 rows and 18 columns) and the 2 regular arrays for the lanthanides and actinides will each have capacities of 15.

Note: although it may be helpful, no previous knowledge of chemistry or the periodic table is required to complete this assignment.

3. Provided files

You are given a text file, elements.txt, which contains all the elements of the periodic table including all the data you need for this assignment and much more. A snippet of this text file is shown below in Figure 3. Most of the data in the file can be ignored as it contains many properties that you will not need to use. The first line of the file is a header which describes the properties of each of the columns. The only columns that you do need to use for this assignment are: AtomicNumber, Element, Symbol, AtomicMass, Period, Group, Phase, Metal, Nonmetal, and Metalloid. If you wish to use other columns, you may do so but you should **not** create additional instance variables in any class beyond what is described in Section 4 (Classes to Implement). There are a couple important notes to keep in mind about this text file:

- Elements in the lanthanides and actinides do not have a value (empty string) for the Group column in the file but they do have a period of 6 and 7 respectively (6 for lanthanides and 7 for actinides).
- Elements must be labelled as either a metal, non-metal, or metalloid. In this file, there are 3 columns for these types. One of the 3 columns will contain the word "yes" to indicate that the element is of that type. The other 2 columns will be blank (empty string). You must determine the type of each element by finding which of those 3 columns contains "yes".

```
1 AtomicNumber,Element,Symbol,AtomicMass,NumberofNeutrons,NumberofProtons,NumberofElectrons,Period,Group,Phase,Radi
  oactive,Natural,Metal,Nonmetal,Metalloid,Type,AtomicRadius,Electronegativity,FirstIonization,Density,MeltingPoint
  ,BoilingPoint,NumberOfIsotopes,Discoverer,Year,SpecificHeat,NumberofShells,NumberofValence
2 1,Hydrogen,H,1.007,0,1,1,1,1,gas,,yes,,yes,,Nonmetal,0.79,2.2,13.5984,8.99E-05,14.175,20.28,3,Cavendish,1766,14.3
  04,1,1
3 2,Helium,He,4.002,2,2,2,1,18,gas,,yes,,yes,,Noble Gas,0.49,,24.5874,1.79E-04,,4.22,5,Janssen,1868,5.193,1,
4 3,Lithium,Li,6.941,4,3,3,2,1,solid,,yes,yes,,,Alkali
  Metal,2.1,0.98,5.3917,5.34E-01,453.85,1615,5,Arfvedson,1817,3.582,2,1
5 4,Beryllium,Be,9.012,5,4,4,2,2,solid,,yes,yes,,,Alkaline Earth
  Metal,1.4,1.57,9.3227,1.85E+00,1560.15,2742,6,Vaulquelin,1798,1.825,2,2
```

Figure 3. Snippet of the first few lines of elements.txt.

You are also given a class, TextFileReader.java, to read in text files such as elements.txt. Ensure that this file is in the src folder of your assignment project folder. The elements.txt file will likely need to be saved **outside** of the src folder, just in the root folder for the project. However, some IDE configurations require it to be saved in a different location so you may have to move it around if it doesn't work when it's saved in the project root folder.

To use the TextFileReader class, start by declaring and initializing an object of the class and sending in the file name as a parameter to its constructor, i.e.

```
TextFileReader fr = new TextFileReader(filename);
```

The easiest way to load in all the contents of a text file is by using a while loop with the endOfFile method from this class, i.e.

```
while (!fr.endOfFile()) {

    ...

}
```

Within the loop, you can get a single line from the text file using the readString() method, i.e.

```
String line = fr.readString();
```

There is a built-in String method split(delimiter) to split a String into a String array based on the delimiter (separator symbol). This will be helpful to break up each text file line into an array of each of the individual property values. You can then use indexes based on the columns you need to use, i.e. index 0 is the atomic number and index 1 is the element name, etc.

Note that all values will be Strings so you will have to ensure non-String values are transformed to the correct type. Java has built-in methods to help with this:

`Integer.parseInt(st);`	to convert st to an int
`Float.parseFloat(st);`	to convert st to a float

Element objects can be created within the loop once all the required parameter values have been properly extracted from the line in the text file.

After all the text file contents have been read in (i.e. after the loop completes), you should close the file, i.e.

```
fr.close();
```

Do **not** edit either of the provided files.

4. Classes to Implement

For this assignment, you must implement three Java classes: **Element**, **PeriodicTable**, and **Compound**. Follow the guidelines for each one below.

Assignment 1

CS 1027
Computer Science Fundamentals II

In these classes, you may implement more private (helper) methods if you want. However, you may not implement more public methods **except** public static void main(String[] args) for testing purposes (this is allowed and encouraged).

You may **not** add instance variables other than the ones specified in these instructions nor change the variable types or accessibility (i.e. making a variable public when it should be private). Penalties will be applied if you implement additional instance variables or change the variable types or modifiers from what is described here.

You may **not** import any Java libraries such as java.util.Arrays or java.util.ArrayList. Doing so will result in strict penalties.

1) Element.java

The Element class represents a single chemical element from the periodic table. The information needed to create objects of this class is contained in the elements.txt file.

This class must have the following private instance variables (no additional global variables are allowed):

- atomicNo (int) i.e. 8
- atomicWeight (float) i.e. 15.999
- symbol (String) i.e. O
- name (String) i.e. Oxygen
- state (String) i.e. gas
- type (String) i.e. Nonmetal

This class must the following methods (no additional methods, other than private helper methods, are allowed):

- Constructor
  - public Element(int *num*, float *wt*, String *sym*, String *nm*, String *st*, String *ty*)
  - Initialize each of the instance variables with the corresponding parameter values
- Getters
  - getAtomicNo(), getAtomicWeight(), getSymbol(), getName(), getState(), getType()
- Setters
  - setName(), setState(), setType()
- public String toString()
  - Return a string representing this element in the format: symbol (name)
    i.e. Fe (Iron)
- public boolean equals(Element other)
  - Determine if this and other are considered the same element based on the atomic number, i.e. if they have the same atomic number, they are considered equal; otherwise they are considered different.

2) PeriodicTable.java

The PeriodicTable class represents the entire periodic table containing all the Element objects from the elements.txt file.

This class must have the following private instance variables (no additional global variables are allowed):

- mainTable (Element[][])
- lanthanides (Element[])
- actinides (Element[])

This class must the following methods (no additional methods, other than private helper methods, are allowed):

- Constructor
  - public PeriodicTable (String *filename*)
  - Initialize each of the 3 arrays with the correct capacities
  - Call loadData() (see description below) to populate the 3 arrays
- public void loadData(String *filename*)
  - Read in the data from the given filename (do not hardcode "elements.txt") using TextFileReader and populate the 3 arrays with the corresponding Element objects
- public String toString()
  - Return a string containing the entire periodic table formatted in a specific style:
    - Each element is shown with its symbol (i.e. Cl) only
    - Each element's symbol must take exactly 4 characters with the symbol letter(s) on the right side of those 4 characters, i.e. 2-3 spaces will be on the left side of the symbol. This will create a clean alignment in the table.
    - Any null cells in the arrays must be shown with 4 spaces so that the structure of the table is maintained including all gaps.
    - The 2D main table must be shown first, followed by a blank line, followed by the lanthanides and then the actinides.
    - When printing out the string returned from this method, it should produce output formatted as shown in Figure 4.

```
  H                                                                   He
 Li  Be                                           B   C   N   O   F  Ne
 Na  Mg                                          Al  Si   P   S  Cl  Ar
  K  Ca  Sc  Ti   V  Cr  Mn  Fe  Co  Ni  Cu  Zn  Ga  Ge  As  Se  Br  Kr
 Rb  Sr   Y  Zr  Nb  Mo  Tc  Ru  Rh  Pd  Ag  Cd  In  Sn  Sb  Te   I  Xe
 Cs  Ba      Hf  Ta   W  Re  Os  Ir  Pt  Au  Hg  Tl  Pb  Bi  Po  At  Rn
 Fr  Ra      Rf  Db  Sg  Bh  Hs  Mt  Ds  Rg  Cn  Nh  Fl  Mc  Lv  Ts  Og

             La  Ce  Pr  Nd  Pm  Sm  Eu  Gd  Tb  Dy  Ho  Er  Tm  Yb  Lu
             Ac  Th  Pa   U  Np  Pu  Am  Cm  Bk  Cf  Es  Fm  Md  No  Lr
```

Figure 4. The expected output from the toString() method in the PeriodicTable class.

- public Element getElement(String *sym*)
  - Return the Element object that corresponds to the given symbol (i.e. if *sym* is "Cl" you must return the object representing Chlorine).
  - The element may be in any of the 3 arrays so ensure that continue checking each until you find the correct element.
  - If you search all 3 arrays and do not find an element with the given symbol, return null to indicate that no such element exists in the periodic table.
- public Element[] getPeriod(int *per*)
  - Return an array containing the elements from the given period index (note that the periods begin at 1, not 0).
  - For this method, the array must only contain the Elements from the period, there cannot be any gaps or null cells in the array (unlike in the instance variable arrays). For example, the array for period 1 must have a capacity of 2 and contain the Hydrogen Element object at index 0 and the Helium Element object at index 1.
  - If the given period is invalid (i.e. <= 0 or >= 8), return null.
- public Element[] getGroup(int *grp*)
  - Return an array containing the elements from the given group index (note that the groups begin at 1, not 0).
  - For this method, the array must only contain the Elements from the group, there cannot be any gaps or null cells in the array (unlike in the instance variable arrays).
  - If the given group is invalid (i.e. <= 0 or >= 19), return null.
- public Element[] getLanthanides()
  - Return the lanthanides array
- public Element[] getActinides()
  - Return the actinides array

3) Compound.java

The Compound class represents a compound consisting of 2 or more Element objects in some quantities. For example, H_2O is a compound consisting of 2 Hydrogen atoms and 1 Oxygen atom.

This class must have the following private instance variables (no additional global variables are allowed):

- elements (Element[])
- elementCount (int[])

This class must the following methods (no additional methods, other than private helper methods, are allowed):

- Constructor
  - public Compound(PeriodicTable *table*, String[][] *compoundArray*)
  - The *table* parameter must be an initialized PeriodicTable object
  - The *compoundArray* object must be a 2D array of Strings. Each String array within the 2D array represents one Element in the Compound and must contain either 1 or 2 values as explained below:
    - The first String of each String array must be an Element's symbol (i.e. Cl).
    - If a String array has 2 values, the second String in the array is an integer (but within quotation marks so it is a String) representing the number of atoms of that Element in the compound.
    - If a String array does not have a 2nd value, assume that there is 1 atom of that Element in the compound.
  - Initialize and populate the two instance variables with the corresponding values (note that the elements instance variable stores Element objects so you must obtain the Element object that corresponds to the given symbol).
  - Since this process may be confusing in writing, please see the following examples to help illustrate what is meant here.
  - The following structure represents an H_2O compound as a 2D array containing 2 String arrays. The first String array contains "H" (Hydrogen) and "2" (meaning there are 2 atoms of Hydrogen in the overall compound. The second String array contains "O" (Oxygen) and "1" (there is 1 atom of Oxygen in the compound).
  - {
    { "H", "2" },
    { "O", "1" }
    }
  - Equivalently, the following array would produce the same results. Notice the second array only contains "O" (Oxygen). Since there is no second String in that array, we use the default value of 1 (1 atom of Oxygen in the compound).
    {
    { "H", "2" },
    { "O" }
    }
- public String getBondType()
  - If there are exactly 2 Element objects in this compound, return the bond type (ensure the spelling is exact and all lowercase) based on the following rules:
    - If one Element is a metal and one is a non-metal, return "ionic"
    - If one Element is a metal and one is a metalloid, return "covalent"
    - If both Elements are non-metals, return "covalent"
    - For any other cases not covered above, return null
  - If there are not exactly 2 Element objects in this compound, return null.
  - *Note that we are simplifying this by only considering compounds with exactly 2 Elements and by not examining all the properties of the Elements which, in reality, could create different types of bonds than listed here.*
- public String toString()

- Return a string containing each of the Element names and the number of atoms of each in the compound, each Element on a separate line as shown in Figure 5.

```
Hydrogen: 2
Oxygen: 1
```

Figure 5. The expected output from the toString() method in the Compound class for a compound of H_2O as an example.

5. Marking Notes

Functional Specifications

- Does the program behave according to specifications?
- Does it produce the correct output and pass all tests?
- Are the classes implemented properly?
- Does the code run properly on Gradescope (even if it runs on Eclipse, it is **up to you** to ensure it works on Gradescope to get the test marks)
- Does the program produce compilation or run-time errors on Gradescope?
- Does the program fail to follow the instructions (i.e. changing variable types, etc.)

Non-Functional Specifications

- Are there comments throughout the code (Javadocs or other comments)?
- Are the variables and methods given appropriate, meaningful names?
- Is the code clean and readable with proper indenting and white-space?
- Is the code consistent regarding formatting and naming conventions?
- Submission errors (i.e. missing files, too many files, etc.) will receive a penalty.
- Including a "package" line at the top of a file will receive a penalty.

Remember **you must do** all the work on your own. **Do not copy** or even look at the work of another student. All submitted code will be run through similarity-detection software. The typical penalty for cases of academic dishonesty on an assignment is a mark of 0 on the assignment and a report sent to the Dean's Office. The following are all forms of academic dishonesty:

- Sharing/sending/showing your code to one or more peers
- Posting your code online (including repositories such as GitHub)
- Looking at another student's code
- Leaving your computer unattended in a public location (such as a library or classroom)

- Using ChatGPT or other generative AI platforms to generate any amount of code or comments for you
- Paying someone to write code for you
- Using code you find online or in a textbook

Submission (due Tuesday, February 4 at 11:55 pm)

Assignments must be submitted to Gradescope, not on OWL. If you are new to this platform, see these instructions on submitting on Gradescope.

Rules

- Please only submit the files specified below.
- Do not attach other files even if they were part of the assignment.
- Do not upload the .class files! Penalties will be applied for this.
- Submit the assignment on time. Late submissions will result in late coupons being applied while they last. Once they are all used, late submissions will receive a penalty of 10% per late day. Submissions will not be accepted more than 3 days after the due date regardless of the number of late coupons you have.
- Forgetting to submit is not a valid excuse for submitting late.
- Submissions must be done through Gradescope. **If your code runs on Eclipse but not on Gradescope, you will NOT get the marks! Make sure it works on Gradescope to get these marks.**
- You are expected to perform your own tests to ensure the correctness of your program.
- Assignment files are NOT to be emailed to the instructor(s) or TA(s). They will not be marked if sent by email.
- You may re-submit code as many times as you wish, however, re-submissions after the assignment deadline will be considered late and handled as such.
- Every file you create for this assignment must include a header including your name, student number, email address, the course code, assignment number/name, and the date on which you first created the file (you don't need to update it when you modify the file). i.e. (the formatting doesn't have to be the same as this example)

```
/*
CS 1027B – Assignment 1
Name: Tom Cruise
Student Number: 123456789
Email: topgun@uwo.ca
Created: July 1, 2025
*/
```

Files to submit

- Element.java
- PeriodicTable.java
- Compound.java

6. Grading Criteria

Total Marks: [20]

Functional Specifications:

[1] Compilation (your code must compile on Gradescope)

[15] Passing Tests (they must run and pass on the Gradescope autograder – we will not be manually grading or modifying these grades)

Non-Functional Specifications:

[1] Header information at the top of all classes you create

[1] Code formatting (readability, indentation, descriptive variable names)

[1] Code comments

[1] Consistent naming convention (e.g. camelCase or snake_case).