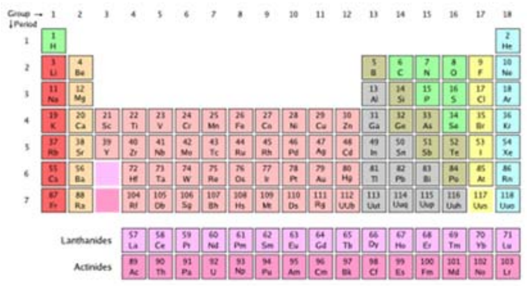
CISC 1115 (Science Section)
Brooklyn College
Professor Langsam



Assignment #4

This assignment will classify chemical elements. Although there are 118 elements listed in the Periodic Table, the program will only work with a smaller subset of them.

A data file, *PeriodicTable.txt*, has been created in the following format:

Atomic number	Name	Atomic weight	Group	Period

You may download the file at: http://eilat.sci.brooklyn.cuny.edu/cisc1115/PeriodicTable.txt . You are to write the following methods:

checkNumber – This method will receive an *atomicNumber* as a parameter and will check whether the *atomicNumber* is a valid number. Valid numbers are the values 1-118. If the number is valid, it should return **true**; otherwise it should return **false**.

readName – This method receives an *atomicNumber* and returns the corresponding *name*. Each time the method is invoked it opens the file and searches for the corresponding entry in the data file.

readAtomicWeight – This method receives an *atomicNumber* and returns the corresponding *atomicWeight*. Each time the method is invoked it opens the file and searches for the corresponding entry in the data file.

readGroup – This method receives an *atomicNumber* and returns the corresponding *group*. Each time the method is invoked it opens the file and searches for the corresponding entry in the data file.

printName – This method receives an element's *name* and prints a heading and the name of that element

isNobleGas – This method receives an integer representing the element's group and returns ***true*** if the element is a noble gas and ***false*** otherwise. Elements in group 18 are noble gasses.

isAlkaliMetal – This method receives an integer representing the element's group and returns ***true*** if the element is an alkali metal and ***false*** otherwise. Elements in group 1 are alkali metals.

isAlkaline – This method receives an integer representing the element's group and returns ***true*** if the element is an alkaline earth metal and ***false*** otherwise. Elements in group 2 are alkaline earth metals.

classify – This method will receive an *atomicNumber* and call the methods, **readName, readAtomicWeight,** and **readGroup,** to obtain that element's relevant information. If the information is not in the *PeriodicTable.txt* data file, it should print the message: "*Element not listed yet."*

In all other cases it calls the methods **printName**, **isNobleGas**, **IsAlkaliMetal**, and **isAlkaline,** in order to print the appropriate information. Note: with the exception of **printName**, all printing is to be done by the **classify** method. Also note, that an element cannot be classified as in more than one group (e.g., it cannot be both a noble gas and also an alkali metal), so you should be using **if…else** statements. If the element is not a noble gas and is not an alkali metal and also is not an alkaline earth metal, the method should print "*Not classified yet*."

Recall that:

number of protons = number of electrons = atomic number

number of neutrons = atomic weight – atomic number

main – The main method is to use a loop to process date contained in an input file. An atomic number is to be read in. The main method should call the method **checkNumber** to check that the atomic number is valid. If the atomic number is valid it should call the method **classify** to print out all the information (as described above) for that element. However, if the element number is not valid, an appropriate message should be printed and go on to the next input.

When all the input has been processed the **main** method should print the total number of input elements processed, and how many inputs were valid and how many inputs were invalid.

Input data:

Create a data file that contains the following entries:

```
2
120
20
19
79
```

Make up 10 more elements that will test your program. Include some invalid data (both too small and too large) and include some elements whose names have not been listed yet as well as some whose types have not been classified yet.

Sample Output:

For the data above your program is to print:

```
Atomic Number: 1 is valid
Name Hydrogen:
Atomic Weight: 1
Group: 1
Alkali Metal
Protons: 1     Neutrons: 0     Electrons: 1


ERROR: Atomic Number: 120 is invalid


Atomic Number: 20 is valid
Name Calcium:
Atomic Weight: 40
Group: 2
Alkaline
Protons: 20    Neutrons: 20   Electrons: 20


Atomic Number: 19 is valid
19 NOT FOUND IN TABLE


Atomic Number: 79 is valid
Name Gold:
Atomic Weight: 197
Group: 11
Not classified yet
Protons: 79    Neutrons: 118 Electrons: 79


Total data in input: 5
Number of correct values in input: 4
Number of errors in input: 1
```

Style:

Each function should have a good comment explaining its role in the program and what parameters it receives. If the function calls another function, the comment should say so. Use meaningful variables and indent your program as described in class. All output is to go to a file. Note that there is no console input required, thus there is no reason to prompt the user.

Programming Note:

1. Since several methods will be printing to the same file, our Printstream object will need to be a *class variable* (global). However, the assignment of this object to the *File* should be done only once in the *main* method and not in other methods.

```
public class Example {

    static PrintStream ps;

    public static void main(String[] args) throws Exception {
            ps = new PrintStream("Results.txt");

                        . . .
```

2. The **readName, readAtomicWeight, and readGroup** methods will perform a sequential search, of the *PeriodicTable.txt* file based on the *atomicNumber* each time they are called. Thus a new (local) Scanner must be declared in each method.

3. A sequential search looks at each item in the file for a match and then returns the desired item. If no item has been founded a dummy item is returned. For example suppose we have a file consisting of *studentID*, *names*, and *GPA* stored in a datafile *Grades.txt*, and we wish to find a student's name based on their *studentID*. We could create a method *find* as follows:

```
public static String find(int studentID) throws Exception {
      int id;
      String name;
      Double gpa;
      Scanner sc = new Scanner(new File("Grades.txt"));

      while (sc.hasNext()) {
            id = sc.nextInt();
            name = sc.next();
            gpa = sc.nextDouble();
            if (studentID == id) {
                return name;
            }
        }
        sc.close();
        return "NOT FOUND";
    } // end find
```

4. Note that all items must be read, as we search, even if we are only interested in finding a single value.

5. Every method which creates a new input stream must throw an exception.