Assignment 1 SU18 CS1027

Due: Tuesday June 26th at 11:55 pm via OWL

In this assignment, you will make a java version of the last assignment from CS1026 SU18.
You will create two classes which together allow a program to store, search, remove and filter country data.

Weight: 5%

This assignment is to be done individually. You may talk to other students on a high level, but all work must be done independently. All assignments will be run through anti-plagiarism software. First offense penalty will be recorded as an act of academic dishonesty, and a mark of -100% will be awarded. You can also help students work on bugs in their code, but the solution is never to "just copy what I did, here look."

Learning Outcomes:

By completing this assignment, you will gain skills related to

- Strings and text files
- Writing your own classes in java
- Testing code
- Using complex data structures (i.e., maps)
- Using java arrays

Task 1. Implement class Country

Instance Variables:

- ☐ name: string
- ☐ population: int (Integer)
- ☐ area: double (Double-length floating point number)
- ☐ continent: String

Methods:

- ☐ Constructor with 4 parameters:
  - String name, int population, double area, String continent
- ☐ Getter Methods:
  - getName, getPopulation, getArea, getContinent,
- ☐ Special Getter: getPopDensity
  - This calculates and returns the population density for the country.
  - Population density is the population divided by the area.
  - *In java, types are very important. If the integer appears first in the expression, it will do integer arithmetic! So, since our population is an integer, and we want to do: population/area, we have a problem. You will need to use something called casting to get java to perform floating-point arithmetic instead, ie.*

```
((double)population)/area
```

- ☐ Setter Methods: setPopulation
- ☐ toString
  - This method is similar to python's __repr__ or __str__ methods.
  - It should return a string that is representative of the object. We will have it print:

```
Name in Continent
Ex: China in Asia
```

Test all your methods in Country.java with the TestCountryClass.java file before moving to the next section. Feel free to create other helper methods in Country.java if desired when implementing Task2.

Task 2. Implement class CountryCatalogue

Instance variables:

- ☐ catalogue: array of Country objects
- ☐ continentMap (equivalent to cDictionary from 1026 Asn3)
  - Dictionaries in Java are called Maps, they "map" one thing, a key, to another thing, the value. The word "map" here is from the concept of mapping in Mathematics (not arithmetic math - more theoretical), so all in all it's a nice pun for this assignment.

Methods:

- ☐ Constructor
  - Like all constructors we will need to initialize all our instance variables. Here we will use two files, the names of which are given as parameters: *continentFileName* and *countryFileName* , both strings.

  Major Steps
  - ☐ Fill the Map:
    - Open the file *continentFileName* and fill *continentMap* .
      - You will need to use the ThingToReadFile class to read in the file.
    - The key is the country name, while the continent is the value.
  - ☐ Fill the Catalogue:
    - Open the file *countryFileName* , then read each line of the file and from that create a country and add it to *catalogue* .
    - You will need to use the "split" method from the String class. For example, if you have the String of a line in the file, you could create an array of the elements in that string that are separated by some delimiter(like a comma):

      ```
      String[] splitLine = line.split(",");
      ```

    - A sample data file has been included data.txt *(Note that both files have headers).*
    - Do not forget to close any files that are opened in your methods.
- ☐ addCountry:
  - given four parameters *countryName* (a string), *countryPopulation* (an integer), *countryArea* (a double), *countryContinent* (a string), first create a country using this information.
  - If there is already a country with the same name, return *false*.
  - Otherwise, there is a new country to add, add the newly created country to *catalogue*. Make sure you add the continent to the *continentMap* .
  - Return *True* after successfully adding the new country.
  - *Hint: you may find the keys of continentMap useful*.
- ☐ deleteCountry:
  - given a parameter *countryName* (a string), if there is a country with this name then it should be removed from the country catalogue entirely (both *continentMap* and *catalogue*).
  - Print a message informing the user whether the country has been successfully removed from the country catalogue or not (was not there in the first place).
  - Do not return anything.
- ☐ findCountry:
  - given a parameter *countryName* (a string), if this country exists then return the country from *catalogue*. If the country does not exist, return *null*.

- [ ] filterCountriesByContinent:
  - given a parameter *continentName* (a string), Print a list containing all the countries that are on continents that are *continentName* , with country names each on their own line.
- [ ] printCountryCatalogue:
  - print the countries of *catalogue* to the screen using the default print (toString) for the Country Class. This method takes no additional parameters.
- [ ] setPopulationOfACountry :
  - Given two parameters *countryName* (a string) and *countryPopulation* (an integer), find the country named *countryName* in the countryCatalogue (if it exists), and set the population of that country to the value *countryPopulation* .
  - Return *true* if the population is updated, and return *false* otherwise.
- [ ] findCountryWithLargestPop :
  - find the country with the largest population, return the name of this country.
  - This method takes no additional parameters.
- [ ] findCountryWithSmallestArea:
  - find the country with the smallest area, return the name of this country.
  - This method takes no additional parameters.
- [ ] filterCountriesByPopDensity:
  - given two parameters *lowerBound* and *upperBound* (both integers), find all the countries (, i.e. country objects) that have a population density that falls within these two numeric bounds inclusively.
  - Print a list that contains all of these countries, with country names each on their own line.
- [ ] printMostPopulousContinent :
  - find the continent with the most number of people living in it.
  - Print the name of this continent with its total population. That is, if *mostPopCont* is the name of the continent found above and *popMaxCont* is the total population of this continent, print:

`The most populous continent is mostPopCont, at a population of popMaxCont.`

- [ ] saveCountryCatalogue:
  - given a parameter *filename* (a string), write the country data of the catalogue as specified below to file *filename*. Each entry should be formatted as below, one country per line.
  - Return the number of lines that were written in the file *filename*.

`Format: Name|Continent|Population|PopulationDensity`

`Ex: China|Asia|1200000000|14.56`

Use the Assignment1.java file provided as the interface to your program. The TA will use a similar Assignment1.java to grade your assignment. DO NOT EDIT THIS FILE AT ALL

For output to both the console and file you do not need to format numbers to include commas. (e.g 1000 does NOT need to be written as 1,000). In addition, when given some string to query (e.g. adding a country, finding a country, deleting a country), you do not need to worry about the casing of the words, e.g. it is OK if findCountry returns None if given "canada" although there is a country named "Canada".

You may assume all the data is correct and there are no errors relating to the data file (so don't worry about Exceptions or validating input).

Non-functional Specifications:

1. **Include a block comment at the top of each file you write identifying yourself and describing the class.**
2. **Use line comments to describe major steps of algorithms or key portions/variables in the code.**

3. **Use coding conventions and good programming techniques, for example:**
    1. **Meaningful variable names**
    2. **Conventions for naming variables and constants**
    3. **Use of constants where appropriate**
    4. **Readability: indentation, white space, consistency**

What You Will Be Marked On:

- Functional specifications:
- Does the program behave according to specifications?
- Does it run with the main program provided in TestCountry.java and Assignment1.java?
- Are your classes created properly?
- Are you using appropriate data structures?
- Is the output according to specifications?
- Non-functional specifications: as described above

Submission

Submit your **Country.java and CountryCatalogue.java** file via OWL as attachments
Double check that they are NOT the .class files
Do not submit any other files, and do not compress (zip, archive, etc.) your files.

FAQ

- How do I do *X* ?
    - Google it first and see if that helps. Ask us if not.
- I'm getting error *X*. What does it mean?
    - Google the entire error message, it might give you some clues. Ask us if not.
- Do I need to write comments?
    - Yes – though the amount you need is really dependent on how obvious the code itself is
- If I submit it at 5:05pm, you'll still mark it, right? I mean, commmmon!
    - 5pm means just that. Expect that submitting code to OWL will take 10-15 minutes. It's not the fastest site, and your connection may be slow when uploading.
- OWL was totally broken, it's not my fault it's late.
    - This is easy to check. If OWL is actually down, email your code to me so we have a timestamp, and then upload the same files to OWL as usual once the site is functional so it is easier for the TAs. We will check that the files match.
- "I accidentally submitted the wrong code." or "I accidentally submitted the .class file instead of the .java file". "Here is the *right* code, but it's late. But you can see that I submitted the *wrong* code on time! You'll still accept it, right?"
    - No. Be careful to double check when you submit.