CS1027: Assignment 1

Due: Sunday, October 8th, 11:55pm.
Weight: 9%

Purpose:
To gain experience with

- Java
- Fixed length arrays

In this assignment you will create a complete program that uses classes to store, search, sort, remove, and filter country data. The two major tasks are outlined below.

Provided With:

You are given a file called *Assignment1.java* that contains a main method. Use this class to run the code you write. Your code **must** work with this file; you cannot alter this file as a means of making your code work. The marker will use a file similar to this to test the functionality of your code.

You are also provided two files called *ThingToWriteFile.java* and *ThingToReadFile.java*. These classes are provided to make fileIO easier. Have a look at the code to understand how to use these objects.

Two text files are included. One contains the country information (*data.txt*) and the other contains the country/continent information (*continent.txt*). **NOTE:** Both of these files have a header line!

On the last page of this assignment you are provided with an example output. Reference this for text formatting.

You may assume all data is correct and that there are no errors in the data files (don't worry about exceptions, or validating inputs, etc.).

Stuff To Do:

1. **Implement a *Country* class:**

    - The purpose of this object is to represent a country. This object will store some important information about the country and will have a small number of simple methods.

    - This class will contain variables for the country *name*, *population*, *area*, and *continent* in which it exists.

    - You will need to write a constructor for the object. This constructor should take a name, population, area, and continent as parameters.

    - Write getter methods for the four instance variables (*getName, etc.)*.

- Write a method called *getPopDensity* to return the country's population density (population/area).

- Write a single setter method for the population instance variable (*setPopulation).*

- Write a method called *writeToFile* which takes a *ThingToWriteFile* object as a parameter. This method must write some country details to the provided *ThingToWriteFile* object provided. Below is an example of how to print the details to file (commas are important):

    i. NAME, CONTINENT, POPULATION, POPULATION DENSITY

    ii. Ex: Canada, North America, 34207000, 3.428881310807587

- Write a method *printCountryDetails* that prints some country details to the user:

    i. NAME is located in CONTINENT has a population of POPULATION an area of AREA and has a population density of GETPOPDENSITY

    ii. Ex: Canada is located in North America has a population of 34207000, an area of 9976140.0, and has a population density of 3.428881310807587

- Write a *toString* method which simply returns a string in this format (include a new line character at the end):

    i. NAME in CONTINENT

    ii. Canada in North America

Test all your classes and methods before moving to the next section. Feel free to create other helper methods if necessary.

2. **Implement a *CountryCatalogue* class:**

    - The class should have a few important instance variables:

        i. A final int called DEFAULT_SIZE which is equal to 5. This variable will be used when the catalogue array is instantiated.

        ii. A final int called NOT_FOUND which is equal to -1. This will be used as a flag for some of our methods.

        iii. A country ***array*** called *catalogue* which will store a bunch of countries.

        iv. An int which stores the number of countries currently in the catalogue

v. A *set* of <Strings> called *continents*. This set will store the continents. Don't know how to create a set in java? Look up the java docs, or just Google "Java sets". This set will make one of the below methods easier to implement.

vi. A <String, String> Map (dictionary) called *cDict* which will map a country name to a continent. This will be used to make some of the below methods easier to implement.

- Write the constructor (this gets a little complex). The constructor should:

  i. Take two strings as parameters. One string will be the name of the file containing the country information, and the other will be the name of the file containing the continent information.

  ii. Instantiate the instance variables.

  iii. Read the country file (with a *ThingToReadFile* object) one line at a time, parse the text, create a *Country* object based on the text, and then add the *Country* to the catalogue with a method called *addCatalogue* (more on this below). Be sure to close the file when done.

  iv. Read the continent file (with a *ThingToReadFile* object) one line at a time, parse the text, and then add the details to *cDict* and *continents*. Be sure to close the file when done.

- Write a private method called *addCatalogue* which takes a country as a parameter. This method only adds the provided country to the catalogue. However, we may have a problem, the *catalogue* may be full! For this method, don't worry about adding duplicate countries.

- Write a private method called *expandCapacity* that will double the size of the catalogue array. This method can be called by *addCatalogue* if there is no more room for a new county to be added.

- Write a method called *addCountry* which takes a country as a parameter. This method should add the provided country object to the *catalogue* and update the *continents* set and *cDict* map. This method can call *addCatalogue*. For this method, don't worry about adding duplicate countries.

- Write a method *getCountry* that takes an index as a parameter and returns a *Country* object from the indexed location in the *catalogue*. This method should return *null* if the index provided is inadmissible.

- Write a method *printCountryCatalogue* which simply calls the *toString* method for each *Country* currently in the *catalogue* instance variable.

- Write a method *filterCountriesByContinent* which will print out all the countries from a specified continent (just use the county's *toString* method). This method will take a String representing the continent we want the countries from.

- Write a method *searchCatalogue* which takes a String of a name of a country we want to find. This method will return an int representing the index of the found country in *catalogue*. If the item is not in the list, return NOT_FOUND. Print some sort of notification to the user if the country was not found.

- Write a method *removeCountry* which takes a String of a name of a country we want to remove from the *catalogue*. This method can use *searchCatalogue*. Print some notification to the user whether or not the item was successfully removed or not.

- Write a method *setPopulationOfACountry* which takes a String representing the name of a country we want to alter and an int representing the new population. This method can call *searchCatalogue*. Print some notification to the user whether or not the item was successfully altered or not.

- Write a method *saveCountryCatalogue* which will write the *catalogue*'s county's to file. This method takes a String representing the name of the file we want to save the details to. This method can simply call the *Country*'s *writeToFile* method.

- Write a method *findCountryWithLargestPop* which will return the index location of the country with the largest population currently in the *catalogue*.

- Write a method *findCountryWithSmallestArea* which will return the index location of the country with the smallest area currently in the *catalogue*.

- Write a method *printCountriesFilterDensity* which prints out some country details if the country lies within some specified population density range. This method takes two integers specifying the range inclusively (low and high). This method will have some additional bookkeeping. See the example output provided below for a demonstration of how this output should look.

- Write a method *findMostPopulousContinent* which prints out the continent, and the total population of that continent (based only on the countries in the catalogue) of the continent with the largest population. See the example output provided below for a demonstration of how this output should look.

Test all your classes and methods. Feel free to create other helper methods if necessary.

FAQ:

- **Q:** How do I compare strings? Can I just use == ?
    - **A:** NO! DO NOT USE ==!!!!!!!!!! Comparing Strings in Java is… very weird actually (and actually the subject of some debate). Use a method called *equals* instead. For example:

        - SomeString.equals(SomeOtherString) will be true if the Strings are the same.

- **Q:** Why can't I use == then?
    - **A:** It's complicated. I'll go into this later in the course.

- **Q:** I don't know how to do X?
    - **A:** Try going to www.google.ca and then typing X into the big text box.

- **Q:** How do we use Maps (dictionaries) in Java?
    - **A:** The idea is similar to python, but there are some Java-isms here. First, check out the java docs for Maps (https://docs.oracle.com/javase/7/docs/api/java/util/Map.html). You'll need to *import* the library they are defined in. Also, try looking up examples of people using them. Instantiating them is a little weird (more details on this later in the course, for now you'll have to just type the *magic code*) (**HINT** for instantiation: *hashmap)*. Iterating through them is different too; try Googling how to iterate through a map in java.

- **Q:** How do we use Sets in Java?
    - **A:** Basically exactly the same answer above, but the java docs are different (https://docs.oracle.com/javase/7/docs/api/java/util/Set.html)

- **Q:** How do I parse a String in Java?
    - **A:** Try looking up "splitting a string in java" with Google.

- **Q:** I swear I did everything right, but for some reason my files won't open!
    - **A:** This isn't uncommon.
        - If you're using eclipse, try putting the .txt files in the project directory (the same folder containing the *bin* folder, *src* folder.
        - If you're running from command line, try putting the files in the *src* folder.

Non-functional Specifications:

1. Include brief comments in your code identifying yourself, describing the program, and describing key portions of the code.
2. Assignments are to be done individually and must be your own work. Software may be used to detect cheating.
3. Use Java coding conventions and good programming techniques, for example:
   i. Meaningful variable names
   ii. Conventions for naming variables and constants
   iii. Use of constants where appropriate
   iv. Readability: indentation, white space, consistency

Submit *Country.java* and *CountryCatalogue.java*. to OWL. Make sure you attach your .java files to your assignment; **DO NOT** put the code inline in the textbox. **DO NOT SUBMIT YOUR *.class* FILES. IF YOU DO THIS, AND DO NOT ATTACH YOUR *.java* FILES, YOU WILL RECEIVE A MARK OF ZERO!**

What You Will Be Marked On:

1. Functional specifications:
   - Does the program behave according to specifications?
   - Does it run with the main program provided?
   - Are your classes created properly?
   - Are you using appropriate data structures?
   - Is the output according to specifications?
2. Non-functional specifications: as described above
3. Assignment submission: via OWL assignment submission

Example output from provided Main.java:

Canada is located in North America has a population of 34207000, an area of 9976140.0, and has a population density of 3.428881310807587

Target country not in catalogue.

England is located in Europe has a population of 54316600, an area of 130279.0, and has a population density of 416.9252143476692

Country "England" removed successfully.

Target country not in catalogue.

Country Catalogue:
China in Asia

United States of America in North America

Brazil in South America

Japan in Asia

Canada in North America

Indonesia in Asia

Nigeria in Africa

Mexico in North America

Egypt in Africa

France in Europe

Italy in Europe

South Africa in Africa

South Korea in Asia

Colombia in South America

Zambia in Africa

Ghana in Africa

Country with the largest population: China is located in Asia has a population of 1339190000, an area of 9596960.0, and has a population density of 139.5431469965489

Country with the smallest area: South Korea is located in Asia has a population of 50503933, an area of 98076.92, and has a population density of 514.9420781158299

Countries in North America:
United States of America
Canada
Mexico

Countries with a population density between 0 and 25:
Brazil in South America
has a population density of 22.716728745947616.

Canada in North America
has a population density of 3.428881310807587.

Ghana in Africa
has a population density of 0.21889705799512446.

Continent with the largest population: Asia, with 1777655033,