**CS1027: Assignment 1**

**Due: February 2, 11:55pm.**

**Weight: 10%**

**Purpose:**

To gain experience with

* Java
* Fixed length arrays

**Task:**

You will recreate your assignment 4 from 1026 in Java. In this assignment you will create a complete Java 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. Two other test files *TestCountry.java* and *TestCountryCatalogue.java* are also posted in the course’s website.

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 described below.

* + This class will contain private instance variables for the country *String* *name,*

*int population, double area,* and *String* *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.

This class will have the following methods, all of which must be public.

* + Write getter methods for the four instance variables: *String* *getName(),*

*int getPopulation(), double getArea(), and String getContinent()*.

* + Write a method called *double* *getPopDensity()* to return the country’s population density (population/area).
  + Write a setter method *setPopulation(int pop)* for changing the value of the *population* instance variable*.*
  + Write a setter method *setContinent(String cont)* for changing the value of the continent instance variable.
  + Write a method called *writeToFile(ThingToWriteFile out)*  which takes a *ThingToWriteFile* object as a parameter. This method must write the name, continent, population, and population density of the Country object (separated by commas) to the provided *ThingToWriteFile* object. Below is an example of how to print the details to file (commas are important):

Canada, North America, 34207000, 3.428881310807587

* + Write a method *printCountryDetails()* that prints on the screen the following details of the Country object:

NAME is located in CONTINENT has a population of POPULATION, an area of AREA, and has a population density of POPDENSITY

For example: 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:“NAME in CONTINENT”

For example: “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.

1. **Implement a *CountryCatalogue* class:**

The class should have the following *private* instance variables:

* + 1. A *final int* called DEFAULT\_SIZE which is equal to 5. This variable will be used when the catalogue array is instantiated.
    2. A *final int* called NOT\_FOUND which is equal to -1. This will be used as a flag for some of our methods.
    3. A C*ountry* *array* called *catalogue* which will store a bunch of countries. When initialized this array will have size equl to DEFAULT\_SIZE.
    4. An *int* called numCountries which stores the number of countries currently in the catalogue. When initialized this variable will have value 0.
  + Write two constructors. The first constructor should:
    1. 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.
    2. Initialize the instance variables.
    3. Read the country file (with a *ThingToReadFile* object) one line at a time, parse the text, create a *Country* object based on the text (the continent for this Country object will be “”), and then add the *Country* to the catalogue with a method called *addCountry* (described below). Be sure to close the file when done.
    4. Read the continent file (with a *ThingToReadFile* object) one line at a time, parse the text to get a country and continent, and then change thecontinent of that country in the catalogue with the method *setContinent* (described below). Be sure to close the file when done.

The second constructor does not take any parameters and it just initializes the instance variables.

* + Write a public method called *addCountry(Country cntry)* which takes a country as a parameter. This method adds the provided country to the catalogue array. The first time that method *addCountry* is invoked, the corresponding country is stored in index 0 of the array. The second country added is stored in index 1 and so on. The value of instance variable *numCountries* increases every time that a country is added to the catalogue.

However, if the catalogue is full, this happens when every entry of array *catalogue* stores a Country object, a new array of size equal to 2 times the size of *catalogue* needs to be created and the information from the old array needs to be copied to the new one. The following method performs this task. 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 will be called by *addCountry()* if there is no more room for a new county to be added.
  + Write a public method *Country* *getCountry(int index)* that takes an index as a parameter and returns a *Country* object from the indexed location in the *catalogue, i.e. catalogue[index]*. This method should return *null* if the index provided is inadmissible.
  + Write a public method *printCountryCatalogue()* which simply calls the *toString()* method for each *Country* currently in the *catalogue* instance variable and prints these strings.
  + Write a public method *filterCountriesByContinent(String continent)* which will print out all the countries from a specified continent (use the Country’s *toString()* method). This method will receive as parameter the name of the continent from which we want the countries.
  + Write a public method *searchCatalogue(String countryName)* which receives as parameter the name of a country and returns an *int* representing the index of the country in *catalogue*. If the item is not in the catalogue, return NOT\_FOUND. Print some sort of notification to the user if the country was not found.
  + Write a public method *removeCountry(String countryName)* which receives as parameter the name of a country and removes it from the *catalogue*. This method can use *searchCatalogue*. Print some notification to the user of whether or not the item was successfully removed.
  + Write a public method *setPopulationOfACountry(String countryName, int pop)* which receives as parameter the name of a country and an integer value and changes the population of that country to the specified value. This method can call *searchCatalogue*. Print some notification to the user of whether or not the item was successfully altered.
  + Write a public method *saveCountryCatalogue(String filename)* which will write the *catalogue*’s content to a file. This method receives as parameter the name of the file. This method calls the *Country*’s *writeToFile()* method for each country in the catalogue. Remember to close the file.
  + Write a public method *findCountryWithLargestPop()* which will return the index location of the country with the largest population currently in the *catalogue*.
  + Write a public method *findCountryWithSmallestArea()* which will return the index location of the country with the smallest area currently in the *catalogue*.
  + Write a public method *printCountriesFilterDensity(int low, int high)* which prints out details of all the countries whose population densities lie within a specified population density range. This method takes two integers specifying the population density range inclusively (low and high). See the example output provided below for a demonstration of how this output should look.
  + Write a public method *findMostPopulousContinent()* which prints out the continent, and the total population of the continent with the largest population (based only on the countries in the catalogue). 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](http://www.google.ca) and then typing X into the big text box.
* **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. **Assignments are to be done individually and must be your own work. Software will be used to detect cheating.**
2. Include comments in your code in javadoc format. Add javadoc comments at the beginning of your classes indicating who the author of the code is and a giving a brief description of the class. Add javadoc comments to methods and instance variables. Read information about javadoc in the second lab for this course.
3. Add comments to explain the meaning of potentially confusing parts of your code. You need to use your own judgment here. If the meaning of a fragment of code is obvious you do not need to add a comment. If someone other than you reading a fragment of code needs to think a few moments to understand how the code works, then write a comment. However, try to avoid comments like these:

*i = 1; // initialize the value of i to 1*

*i = i + 1; // increase the value of i*

*if (i == j) // compare i and j*

1. Use Java coding conventions and good programming techniques, for example:
   1. Use meaningful variable and method names. The name should help understand what a variable is used for or what a method does. Avoid the use of variable names without any meaning, like “xxy”, or names, like “flower”, that do not relate to the intended purpose of the variable or method.
   2. Use consistent conventions for naming variables and names of methods and classes. For example, you might decide that names of classes should start with a capital letter, while names of variables and methods should star with a lower case letter. Names that consist of two or more words like “symbol” and “table” can be combined, for example, using “camelCasing” (the words are concatenated, but the second word starts with a capital letter: symbolTable) or they can be combined using underscores: symbol\_table. However, you need to try to be consistent.
   3. Use consistent notation for naming constants. For example, you can use capital letters to denote constants and names composed of several words can be joined by underscores: TABLE\_SIZE.
   4. Use constants where appropriate.
   5. Readability. Use indentation, tabs, and white spaces in a consistent manner to improve the readability of your code. The body of a *for* loop statement, for example, should have a larger indentation than the statement itself:

*for (int i = 0; i < TABLE\_SIZE; ++i)*

*table[i] = 0;*

Also, positioning of brackets (“{“ and “}”) to delimit blocks of code should be consistent. For example if you put an opening bracket at the end of the header of a method:

*private int findPosition() {*

*int position;*

You should not put the bracket in a separate line for another method:

*private String getName()*

*{*

*return personName;*

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 test programs provided?
* Are your classes implemented properly?
* Are you using appropriate data structures?
* Is the output according to specifications?

1. Non-functional specifications: as described above
2. Assignment submission: via OWL assignment submission

**Example output from provided Main.java:**

Canada has index 4 in the Catalogue

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 has index 16 in the 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 population 1777655033