Homework 7

CSCIE-55

Fall 2019

Fred Evers

# **Homework #7 -- Adventures in Java**

Due: Monday December 16th

Grading: 10%

Last Modified: Monday, 2-Dec-2019

# **Problem 1 Java Streams and Predicates.**

Complete the code for BeerFest.java in the problem2 project. It contains unimplemented code [see TODO comments.]

This is a pretty simple problem asking you to create 2 static Predicates of type Predicate<Beer>, and pass the Predicates as parameters to a method that will use them in calculating its results.

Remember that a Predicate is a ‘test’ that returns a boolean.

The first Predicate will be used to test if items in a stream of Beer originate in a specified country.

The second will be used to test if items in a stream of Beer fall into a specified price range.

Class BeerFest contains a main method. You can run and debug this problem directly in the class. As such, most of the other methods are also static so they are accessible to the main method.

The file in project1 has package structure: ***cscie55.hw7.problem1***

What to Submit

A README describing how you solved the Predicate design, and your project assets including the edited BeerFest file. Please delete all .class files from target directory. Zip the assets into a zip named lastName\_firstName\_project1.zip

# **Problem 2 – Library Checkouts w/ Java 8:**

No VM, no Hadoop. Back to your tried-and-true development environment.

In this problem we will repeat parts of the HW 6 problem 3 exercise, where you parsed the smaller\_seattle.csv file to identify the most popular titles. The ‘driver’ for this problem’s code will be Junit tests that you will create.

Some of you discovered and used Apache common csvParser for HW 6 problem 2. This time we will all use Apache commons csvParser, Java 8 streams and lambda expressions to read the file and generate a list of Checkout items, sorted by item title and accessible via methods you will write.

The assets on BitBucket for problem 1 contain a new pom.xml file that includes the dependency definition for apache common csvParser.

See that project2 has package structure: ***cscie55.hw7.problem2***

* All classes that you add should be in that package.
* Define a class named “Library”
  + Library will contain a Collection<Checkout>
  + Library will contain a private object Library library;
  + Library will have the following methods:
    - public getMostPopular(int limit)
    - public getAuthorsBeginsWith(String s)
    - public getAllPublishedIn(int year)
    - public getFilteredBy(Predicate p)
    - All of the above ‘get’ methods should return List<Checkout>
  + Library will only allow one instance. I.E., it is a ‘singleton’.
    - Enforce the only-one-allowed feature by creating these methods:
      * A private, no argument constructor
      * A public getInstance() method that returns library
* Define a class named “Checkout”.
  + Checkout class will have the following fields and methods:
    - A no argument constructor
    - A constructor to take parameters to populate the fields for the class.
    - Include override methods: toString(), equals() and hashCode().
    - enum Kind {PHYSICAL, DIGITAL, OTHER};
    - String title;
    - String author;
    - int publicationDate;
* Define a class named “Reader” that is responsible for retrieving the data from GitHub, parsing each line using csvParse to create a Checkout object, and adding each one to the Collection<Checkout> in class “Library”
* Define a class named “CheckoutWriter”.
  + CheckoutWriter will have the following methods
    - write(Path filePathAndName, List<Checkout> checkouts)
* Create Junit tests to instantiate your classes and test run each method.
  + In your submitted project, generate 3 output files by calling your Library ‘get’ methods and passing the returned data to the CheckoutWriter’s write() method.

What to Submit

A README describing why you chose the Collection type used, and your complete project including the 3 generated output files. Please delete all .class files from target directory. Zip the assets into a zip named lastName\_firstName\_project2.zip

# **Problem 3 Java, Jackson, Deserialize and Serialization JSON**

* Read Dishes from AmazonS3 bucket
* Convert to list [menu] using Jackson as we did in HW 5 [see unit test testReadStreamToJson() for example of input stream to Dish[] array]
* Implement method partitionByCalorieLimit() to partition into calories above and below an int ‘calorie’ param using java’s Collectors.partitionaingBy() function.
* Implement method getVegetatianMenu() using Streams filters to create a new menu of vegetarian dishes.
* Ensure that the test testGetVegetarianMenu() passes.
* Create functional interface [an interface w/ one abstract method] named ‘Slicer’ in package cscie55.hw7.api
* Make this method generic, with type parameter T for input type; R for return type; V for another type.
* In the Slicer interface, define an abstract method named ‘slice’ such that the method expects input (T, V, V) and returns R
* In the TakeOutShop, declare a variable of Slicer named ‘slicer’, defining the type parameters as: List<Dish> for T, List<Dish> for R and Integer for V.
* In the body of TakeOutShop, implement ‘slicer’ as a method with the following signature: ***List<Dish> slicer(List<Dish> dishes, Integer start, Integer end)***
* In the body of your slicer method, write code that returns a subset of the dishes list, beginning with position ‘start’ and ending with position end.
* Ensure that the test testGenericMethodSlicer() passes.
* Note that the testGenericMethodSlicer() calls the Publish method from HW 5 to a file named “cscie55menuSlice.json”

What to Submit

A README describing your work and your complete project, including the test directory. Make sure file “cscie55menuSlice.json” is included with your assets. Please delete all .class files from target directory. Zip the assets into a zip named lastName\_firstName\_project3.zip

### **Problem 4. Anagrams. [The job interview question…]**

Anagrams are words or sequences of words that use exactly the same letters but in a different order. For instance the words "now" and "won" are anagrams of each other.

Your task in this problem is to identify anagrams in a list of words. Considerations:

Consider each line in the file as a record, i.e. a word or sequence of words that is an anagram candidate.

Ignore white spaces and punctuation in testing for anagrams

Ignore cases, upper vs. lower.

To look for anagrams you need to extract a core representation of the constituent characters of each record from the data file, call it the "character key string.". For example, for the line that reads "Wolfgang Amadeus Mozart" you should generate a pair of strings, the first being the character key string, "aaaadefgglmmnoorstuwz", and the second be the word or phrase on the line. Likewise, when your program eats the line "A famous German waltz god", it should generate the same key string, allowing it to identify the two strings as anagrams of one another. Notice that the letters are in alphabetic order and that each letter appears as many times in the key string as it occurs in the line. Thus, your stream processing should generate a pair of strings,

Use Java 8 streams to:

* read the input lines
* eliminate duplicate lines, and
* generate Key/Value pairs combining the character key string and the line
* collect the results into a map where the key is the character key string and the value is the string concatenation of the anagrams. Output should have the form:
* eeehnoorttw->one two three...three two one

Run your solution against the provided data file in anagram-data.txt and submit the last 20 lines of the output together with your Java code.