Homework #5: File IO; Jackson

Due: Monday, November 18th 11:59 pm Eastern time

Grade: 10%

Last Modified: Tuesday, 5-Nov-2019 15:03:34 EDT

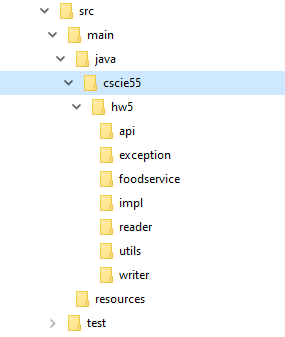
**Note: For this assignment and future assignments you no longer need to generate or include JavaDoc.**

**Specification of Requirements**

# **Overview**

This homework uses the FoodService application to demonstrate several techniques related to Reading and Writing Files.

There are now 7 packages:



1. cscie55.hw5.api
   * Order.java
   * Passenger.java
   * Person.java
   * Shop.java
2. cscie55.hw5.exception
   * ElevatorFullException.java
   * KeyDoesNotFitException
   * NoSuchApartmentException
   * TooManyResidentsException
3. cscie55.hw5.foodservice
   * Chef
   * DeliveryPerson
   * Dish
   * FoodOrder
   * TakeOutShop
4. cscie55.hw5.impl
   * Address
   * Apartment
   * Building
   * Elevator
   * Floor
   * Resident
   * WebBrowser
5. Reader
   * MenuFileReader
6. cscie55.hw5.utils
   * NumUtil
7. Writer
   * MenuWriter

Many of these packages and classes are familiar from HW3 or from in-class exercises and demos. There have been some slight modifications to the classes from versions you have already seen. I recommend NOT using older versions and sticking with the download from HW 5 from BitBucket.

For **cscie55.hw5.reader** we have the following addition:

# MenuFileReader

This class has one method: read().

The read() method takes a String as a parameter. This value is resolved in the body of the method to form the name of a File at the end of a file path.

It is expected that the file that will be read is in JSON format that can be transformed by Jackson libraries into a List of Java Dish objects. Plain Old Java Objects – POJOs.

When it reads a json file correctly, the method returns a List<Dish> object.

For **cscie55.hw5.writer contains the following class:**

# MenuWriter

This class contains one method: publish().

Method publish() takes a String and List<Dish> as parameters.

The String is used as a filename when the method generates its output.

The List<Dish> is a Collection that contains Dish objects.

Note that for our TakeOutShop, the menu is of type List<Dish>

One thing the MenuWriter’s publish method can be used for is generating menus to be used by Shops, like our TakeOutShop.

Another thing it can be used for is to generate receipts. Read on.

For **cscie55.hw4.foodservice** has a number of TODO:’s.

You are only responsible for the ones listed here.

# TakeOutShop

Implement addMenuItemsList method.

The intention here is that the parameter (List<Dish> dishes) will be added to the existing menu, increasing it by an additional dishes.size()

Implement generateReceipt() method.

This method takes a FoodOrder as a parameter, and gets the address.toString() value and food items from FoodOrder to pass as the String parameter to MenuWriter.publish(String filename, List<Dish>).

For **cscie55.hw4.impl** we have the following:

# Address

Implement a toString() method. Note that the value of this method is used to create the name of the generated Receipt in TakeOutShop.generateReceipt(). Consider that the result of the Address’s toString() method will be the first part of a file name.

# TestFoodService

The test code as before is under the src/test branch. The test file is named FoodServiceTest. The unit tests in Homework 5 contain some tests that have been delivered to you before as in-class demos.

Unlike previous assignments, there is some customization for you to do in the unit tests. Those that require your modification are highlighted in yellow below and discussed afterwards.

The new tests listed below are for you to resolve by implementing workable solutions

* testAddMenuItemList()
* testSetNewMenu()
* testFileReader()
* testPublishMenu()
* testGenerateReceipt()
* testLoadExternalMenuFile(){

testAddMenuItemList()

First implement the TakeOutShop’s addMenuItemList() method, and use this test to validate it.

For the later, add 10 menu items following the object creation patterns shown when the original menu is created in the TakeOutShop class. But here, add 10 new Dish items, and use the addMenuItemList to extend the original menu by 10.

testSetNewMenu()

You were asked to implement the setNewMenu method in TakeOutShop. This test uses a random order generator to create a new List<Dish>, and sets the menu in the TakeOutShop.

testFileReader()

Your project comes with a menu.json file that contains a json formatted version of the original takeOutShops menu. This test loads that file and verified that the number of items is the same as the original.

testPublishMenu()

In this test, you must add 10 Dish items, as you did for testAddMenuItemsList(). But this time, you will first set the new menu items as the takeOutShop menu using setNewMenu(). Then publish that menu to json. You must provide a name for the file.

testGeenerateReceipt()

For this test, an address is hard-coded and a random order is created. These are passed to the generateReceipt method that you created in TakeOutShop class. Verify that the generated file is what you expect.

testLoadExternalMenuFile();

For this test, the file that you published defining testPublishMenu() should be retrieved from disk and loaded into the takeOutShop’s menu. You must provide the same filename that you used when you generated the file in testPublishMenu(), and verify that it is loaded. The test itself only expects a menu.size() of 10, as opposed to the default 9.

Be advised that UnitTests do not always run in a predictable sequence. Note that testPublishMenu() must preceed testLoadExternalMenuFile()

# What to Submit

A README file describing all results. Please include your sources, test sources and all generated resource files.