CS 3053
Project – Populated Application
Due Friday 2018.04.13 at 11:00pm.

Overview

In this assignment you will extend your app to load, edit, and save actual data for your theme. In class, we have encountered many different components and principles for combining them in our user interface designs. Out of class, you will build on what you have learned to transform your evolving prototype into a basic yet functional application. You will develop a format for storing theme collections as files, assemble a representative data set in that format, implement reading from and writing to that format, and support opening and saving of files during editing.

<u>In this assignment, **all group tasks come before individual tasks.** Make sure to leave enough time for all of your team members to complete their individual tasks, especially implementation.</u>

Group Tasks

*<u>Complete **all** group tasks before proceeding to individual tasks. Your entire team **must** participate.</u>*

#1: Discuss the aesthetics of your individual Styled Application implementations. Decide which design choices in each to integrate into a unified design for your entire team. <u>Integrate, refactor, clean up, and **thoroughly** document your code in preparation for the final implementation steps.</u> This will be the code base you all start from for the following group <u>and</u> individual tasks. Briefly summarize who contributed which UI components and code improvements to your refined build.

#2: Figuring out how and where to store data is one of the most important steps in app design. Although many apps format their files using XML or JSON, quite a few still use a custom text or binary format. To keep things simple, you will format your app files using the Comma Separated Values (CSV) format. Read about it in Wikipedia and at tools.ietf.org/html/rfc4180.

#3: Discuss how to use the rows in a CSV to store information about each item in a collection in your theme. Define columns to represent the data attributes for each item. Define the name, the data type, and the allowed data values for each attribute. Describe how to convert those data values to and from strings when reading and writing to CSV format. Together, these choices specify a *data schema* for your CSV file format. Document your schema clearly and completely.

Note that it's perfectly fine to use multiple columns to represent a complex data attribute! For example, a geographic location can be stored as a pair of columns, one for latitude from -90º to 90º and one for longitude from -180º to 180º, both of type `double` in memory and written to CSV as a pair of decimal strings formatted like `35.22,-97.44` (for Norman).

#4: Build a representative data set to test the browsing, editing, reading, and writing features of your app. Individually, collect information about 5–10 items within your team's theme, then record each item as a row in a CSV file, according to your data schema. Together, gather your individual CSVs into a single CSV. Briefly describe any significant difficulties you ran into while gathering, recording, and combining data. Discuss how reliable, consistent, and complete your resulting CSV data is as a representative example of possible collections in your theme.

#5: Document any special kinds of data, such as images or other media, that are associated with items in your collection. Although playing media like audio or video in your app is beyond the scope of the project, showing images is relatively easy. For each item in your example file, find a suitable PNG or JPEG image. (Convert them in an image editor if necessary.) Name the image files appropriately and put them into the resources/images directory. Include the image filename as an attribute in your data schema, and add the filenames to the corresponding column of your example CSV. To access the image for a given item in your program, call the Use `Resources.getImage()` to load item images in the same way as you loaded icons for the toolbar. Describe where item images are displayed in your design. If your UI doesn't show item images, add a small `JLabel` with an image icon to your design to demonstrate the possibility.

#6: Write up your work on each of the parts of tasks #1–#5. Compose this together. Be clear, objective, detailed, and thorough, yet succinct. *In grading we will be looking in particular for: a sincere, substantial effort to produce high-quality, well-factored code (#1); a clear and complete specification of a data schema appropriate to your theme (#3); a data set that represents a wide range of reasonable items in your theme, with a clear-eyed assessment of the quality of that data for testing (#4); an appropriate collection of images, correctly named in the example file, with a helpful description of where those images are meant to appear in your UI (#5).*

Your writeup should be between 1.5 and 2.0 single-spaced pages of writing. Start the first page with a few lines stating your team number, name/logo, and list of member names. Use regular paragraphs and standard formatting (12 point font, 1 inch margins, etc.) For the data schema, you may use a table or a well-formatted list to help convey the details more clearly. Attach a screenshot of the UI for your refined build (#1) and a table of your example CSV data (#4), using a full page for each. Refer to them in your writeup appropriately.

To **turn in** your group work, go to the "Group - Populated Application" assignment in Canvas to submit your results as a PDF. Only one team member needs to turn in the group component.

Individual Tasks

All individual tasks must be completed entirely on your own.

#7: Start from a copy of your team's integrated build from task #1. Duplicate the stage6 main() class, call it `Stage7.java`, and modify `build.gradle` to have a new `createScript()` line. Whenever you build, the executable `stage7` should appear in `build/install/base/bin`.

#8: Read about using the Apache Commons CSV library to read and write CSVs:
http://commons.apache.org/proper/commons-csv/
I extended the starter code in the provided `Stage7.java` with an example of loading CSVs. It uses the `CSVFormat` and `CSVParser` classes to load `resource/data/stadiums.csv`. Press the button in the running UI to see the data both in a `JTable` and printed to the console. ***Important:*** *Use the new* `build.gradle` *script, which includes the new library dependency.*

#9: When your app starts, show your main browser/editor frame in the "new" state, i.e., without any data. When the user selects the Open action (using the menu item or otherwise), display a `JFileChooser` for loading from a `.csv` file. If they choose a file that exists, try to read the CSV into an appropriate memory structure. If the read succeeds, populate the UI with the collection information and put the UI into a suitable starting state for browsing. If the read fails for any reason, display a suitable message in `JOptionPane` then return to the empty main window.

#10: When the user selects the Save action (using the menu item or otherwise), display a `JFileChooser` for saving to a `.csv` file. If they choose a file that doesn't already exist, create a `File`, then write the CSV file to it. If they choose a file that does exist, display a suitable error message in `JOptionPane`. **Do not provide an option to overwrite the file.** Don't close the window or exit after the save attempt regardless of the result; just let the user continue working.

#11: When the user selects the Quit action (using the menu item or otherwise), check to see if the user has made any changes to the data since the most recent successful Open. (A simple way to do this is to have a flag that is set to true whenever the user makes a change. Reset the flag after a save.) If the data hasn't changed, simply exit the program. If the data has changed, use a `JOptionPane` to ask the user if they wish to: (1) quit without saving; (2) cancel; or (3) save before quitting. In these cases: (1) simply exit; (2) just let the user keep working; (3) exit if the save succeeds, otherwise use a `JOptionPane` to show a warning message before exiting.

In your implementation, create appropriate new classes to add to your team's refactored code base inside the `edu.ou.cs.hci.stages` package. Document your new code thoroughly and appropriately.

Demonstrate that your app successfully loads from and saves to your team's specified CSV file format. Load the example CSV file (#4), perform a sequence of typical browsing and editing interactions to change the data, take a screenshot of your frame, then save the changed data to a new CSV file. Put the new CSV file in the `Results` subdirectory as `editing.csv`. Trim the screenshot, turn it into a PDF, and put it in `Results` as `editing.pdf`. Write down the interactions you performed between load and save and list them in `Results` as `editing.txt`.

To **turn in** your individual work, *first test your project <u>using Gradle on the command line</u> to make sure it builds and runs as intended.* Run `gradle clean` to reduce the project size. Append your 4x4 to the `project` directory; mine would be `project-weav8417`. Zip the renamed directory. Submit your zip file to the "Individual - Populated Application" assignment in Canvas.