Non Data-Driven GUI Handover Documentation

Team responsible for the work listed below, and their PSU email addresses:

- Darius Holt - dqh5300@psu.edu
- Jaime Seaman - jss5783@psu.edu
- Rishabh Shanbhag - rgs19@psu.edu
- Michael Strizziere - mrs6041@psu.edu

I. Help Documentation

- **Created/worked on by Michael Strizziere and Rishabh Shanbhag**
- **Explanation:** The help documentation contains all of the help contents for users to refer to regarding the contents of the application itself. It has an explanation of the contents of the different menu items, panels, buttons and their functions. You can locate the *project file* under resources > default package > CourseManagement.hnd. You can also locate the *HTML files* generated from the project file under resources > HelpDoc. The main HTML file is CourseManagement.html. In the application itself, the document can be opened by pressing “Help Contents” under the “Help” menu item. This button is located in the jfMain GUI, and the action performed is located in the source code under jmHelpContentsActionPerformed. The current method to open the help documentation is via the HelpFX class. All old methods are commented out.
- **TODO:** Upon starting this course, the help documentation will already be out of date due to new menu items, panels and buttons being added to the application. The first step would be to update the document with whatever new contents need to be added to it. Some contents have been removed, as well, which means they also need to be taken off

of the help documentation. In order to do this, you will have to download the project file listed above and download HelpNDoc at https://www.helpndoc.com/. After the changes have been made, generate the new HTML documents and replace the old ones with the new ones. The generate button will be one of the first buttons listed under the home tab of HelpNDoc. The second step would be to get the ICEBrowser method of opening the help documentation to run, and replace the HelpFX method of opening the help documentation with it. We could not get it to run in the little time we had left at the end of the semester. Professor Bowers can explain this process.

II. SplashScreen

- **Created/worked on by Rishabh Shanbhag and Jaime Seaman.**

- **Explanation:** Providing users with feedback is an important aspect of UI design, and hence we kept that in mind while deciding to build/design the splash screen. The splash screen is used as an introduction page, before the application actually loads up. Hence, acting as an indicator to the user that the application has started to load or is loading up. The splash screen currently displays the application logo, the application name, "© 2017 PSU Berks IST", an inspirational quote, and a loading bar with a matching color scheme. It also consists of a “Tip of the day” functionality, where random quotes and the “quotee” are fetched from a view present in the SQLite database “Today.db3”. The connection to the database is made through the file “DBConnection.java” whereas the rest of the splash screen functionalities are present in “SplashScreen.java”. The splash screen is created in AppControl.java.

- **TODO:** Clean up code (add comments, organize/remove extraneous code, etc.), clean up appearance (e.g., extra long inspirational quotes are cut off, quoter names should be

on a separate line, non-italicized, e.g. "Quote text," --FirstName LastName), loading bar should actually matter (currently hard-coded "loading" wait)/loading label should show loading comments.

III. Docx4j

- **Worked on by:** Darius Holt
- **Description:** Creating and editing report templates using Docx4j.
- **Explanation:** Understanding the basics of Docx4j and how to use the tools was not so much as difficult as it is just a ton of information. Most of what I worked on is in the reports section and templates for the reports. The reports should work as follows: A connection to the database is made between report.(add report type here) and then the information related to that type of report will be added to a template that fits that report. The rest is just saving the report as a docx file, converting to pdf, and creating different types of reports. Currently the only document made should be a syllabus, but it should provide a good layout for how the reports should be done.The biggest issue I had was using the tools Docx4j gave to me, and learning how to put it together. Any questions on what exactly you should be doing with this feel free to email me at the address given at the top of this document.

- **TODO:** Following the layout create the rest of the reports is one task here. The bigger task I feel is creating more customized reports. Reading the comments I will be leaving in reports maybe helpful in regards to this. I will be adding some commented code as to how exactly you can add text to the word document, what is left to figure out from there is how to make them customized at that point.

- **TODO:** Libraries, which ones are needed and which are not, I will be adding a list of libraries that I was going to go through if I finished reports, seeing that I could not, whomever does needs to go through this list and see which can be removed if any.

IV. jfAbout

- Created/worked on by Jaime Seaman.
- **FILE DESCRIPTION/PURPOSE:** Creates an about screen in NetBeans's GUI form designer. Accessed through jfMain's drop-down menus (Help -> About).
- **IMPLEMENTED:** The about screen displays the application name, version number, and one of three TXT files, depending on the radio button selected (rbtnContributors: contributors.txt; rbtnResources: resources.txt; Tools: tools.txt). The loaded text displays as HTML into a text panel, including working web and email links.
    - Contributors lists the people who have worked on the program, past and present, with the teacher's name at the top, and includes anti-spam text in the TXT file itself to prevent spambots from scraping them easily (since all files are publicly displayed in GitHub); each person lists their first and last names, and any methods of contact (0+ emails and/or websites in any order). All files are semi-colon-delimited (i.e., fields are separated by ';').
    - Resources displays all resources (e.g., libraries, images, icons) used in the application. They include the general name for the resource(s), the website URL, the license version, the URL to the full license, a description of what the resource is/what it's used for in the application, and the individual files in the resource.
- **TODO:** The application license should be at the top of resources.TXT so it displays first, probably with the custom splash/application icon and its license below it.

        - (Not that it can really be proven that I wrote this, but here's written permission:) *I, Jaime Seaman, creator of the custom splash/application icon images (description: a stylized lion head with a scroll tied with a tasseled cord in its mouth), permit them to be licensed under the same license as the application itself.*
    - The text should be displayed in the text panel more elegantly, taking advantage of HTML to display as actual lists.
    - Some files would perhaps be better if converted into XML files, especially if they get more complex.

IV. jfReport

Created/worked on by Jaime Seaman.

FILE DESCRIPTION/PURPOSE: Creates a report generator in NetBeans's GUI form designer. Accessed through jfMain's drop-down menus (File -> Generate Report). The report generator displays a list of profiles, a series of combo boxes for selecting Department-Course-Section (i.e., a specific class), and a series of tabs used to customize the template that the report is generated with, fill in the template with data retrieved from the database based on the components and class selected, preview the report, and customize the filepath, filename, and filetype(s) before finally generating the report.

IMPLEMENTED:

- *Profiles:* Partial placeholder functionality has been added.
- *Class:* Department, course, and section can all be selected correctly and without errors. "---" is used to keep any real values from being selected as they're loaded in from the

ResultSet, and looks nicer and more professional than the empty value that cannot be selected after a real value is selected.

- *Template tab:* Components can be added and removed from the list of template components, non-table components can be added and removed from the currently selected table component in the list of template components, the table panel hides/shows itself depending on whether or not a table component is selected.
- *Content tab:* Layout is pretty much finalized, but right now, all components just hold placeholder data. It checks for a valid selected class (department, course, section must all have real values selected) and that the list of template components is not empty; if either set of conditions is false, then the user is returned to the Template tab.
- *Preview tab:* Not yet touched beyond its basic existence, as I was uncertain how to display the report and how to save it so the report would not need to be generated again when actually generating the report in the Generate tab.
- *Generate tab:* Filepath, filename, and filetype pickers all work correctly. However, the report generation functionality is currently completely broken - the placeholder test report generation is still present in the code, but actual basic *real* report generation (or a quick test of just getting data from the database, as I was attempting) was never completed due to misunderstanding how ReportSets work. Furthermore, the loading bar doesn't even show despite being coded to do so, and the progress label that accompanies it only shows the progress completed texts.

TODO:

- *General:* There should be a checkbox somewhere for disabling most/all of the confirmation dialogs, for fast but risky operation (e.g., an accidental misclick wipes out the currently edited template list).

- *Profiles:* Needs to dynamically load into a ListModel the names of profiles from a file.
    - Needs to ask if the user wants to generate a blank profile or copy over the currently loaded one (as in, the one that may or may not have been edited since loading the profile), and then create a new profile in an XML or something.
    - Needs to load in a profile after asking the user if they want to save the current profile's changes (if there are any) first, or just if they want to load a new profile and lose any loaded content items (if there are any).
    - Needs to save the current profile to file, asking if they want to overwrite.
- *Template tab:* The Template tab should probably have a Clear button for the regular list of components and the table components, so it's easy to start from scratch (ask for confirmation first).
    - The components should be drag-and-droppable both in the regular list of components and the table components.
    - There should be static tables and dynamic tables, marked with different braces for differentiating in code, and a way to mark whether a table is a pattern (to be read multiple times until data runs out, like for a course schedule or required resources table in the syllabus) or irregular (for odd tables that might need empty cells or non-repeating patterns).
        - If a pattern, it's possible that there's only 1 row, and the number of rows cannot be increased (disabling spinnerRow for patterns after setting the number of rows to 1 if it's not already; since this can be destructive, this should be activated on clicking Update Table), or perhaps there are also irregular tables.

        - There should be a way to indicate whether or not tables are generated with titles, and a way to customize them (if patterns are 1-row-only, then they can be derived from the columns' components).
- *Content tab:* DefaultListModel should be registered to the associated GUI component, and the DefaultTableModel array registered to the associated GUI component as needed (see the Template tab and associated code for reference).
    - The components in the Template tab should be read to compile a list of what needs to be read into a ResultSet? before actually doing so, so everything can be read into a single ResultSet.
    - Since the template components and content probably need to share a ListModel, the template components should be marked in a way unlikely to be mixed up with an actual content item when going through the list to make sure it's been filled out (e.g., double braces like [[COURSE_NAME]]).
- *Preview tab:* The Preview tab should kick the user back to the Content tab if Content is not complete (all components replaced with actual content, ignoring empty cells and things like PAGEBREAKs and possibly short, other uninformative fields like CLASS_NUMBER (which is stuff like 451)); the generate button should probably just not allow generation instead (so users can customize filepath and such ahead of time). (If this triggers the check for "is there a class selected and a non-empty component template?", then it should check for a filled template, though it's possible to have a filled-but-technically-correct (outdated) template in through specific circumstances by accident).
- *Generation tab:* There should be a loading bar that reflects the state of file generation, either overall (increasing progress as files are created and then saved) or more

specifically (starting from 0% and increasing to 100% as files are created, starting from 0% as the DOCX is created, and then starting from 0% as the PDF is created).