1. List the features that were implemented (table with ID and title).

UR-001	User can generate a resume through an interactive GUI
UR-002	User can input all of his/her personal information into individual fields
UR-003	User can open a resume that he/she created previously
UR-004	User can save his/her resume in an easy to import format
UR-005	User can import his/her resume as a word document (.docx)
UR-006	Users can add any number of arbitrary sections
UR-007	Users can add or delete items from a section
FR-001	Code automatically generates a resume given inputted user information
FR-002	Several blank fields will be provided upon creation of a new resume
FR-003	Resume sections are highly customizable, including renaming sections
FR-004	Program exports resumes to one attractive format (.docx)
NFR-001	The ResumeBuilder does not lose data when importing or exporting resumes
NFR-003	The ResumeBuilder GUI is compatible with assistive technology such as screen readers
UC-01	Users can create new resumes from scratch
UC-02	Users can create new resumes with their personal info prepopulated, if the user has previously provided that info
UC-03	User can edit an existing resume to make any necessary modifications

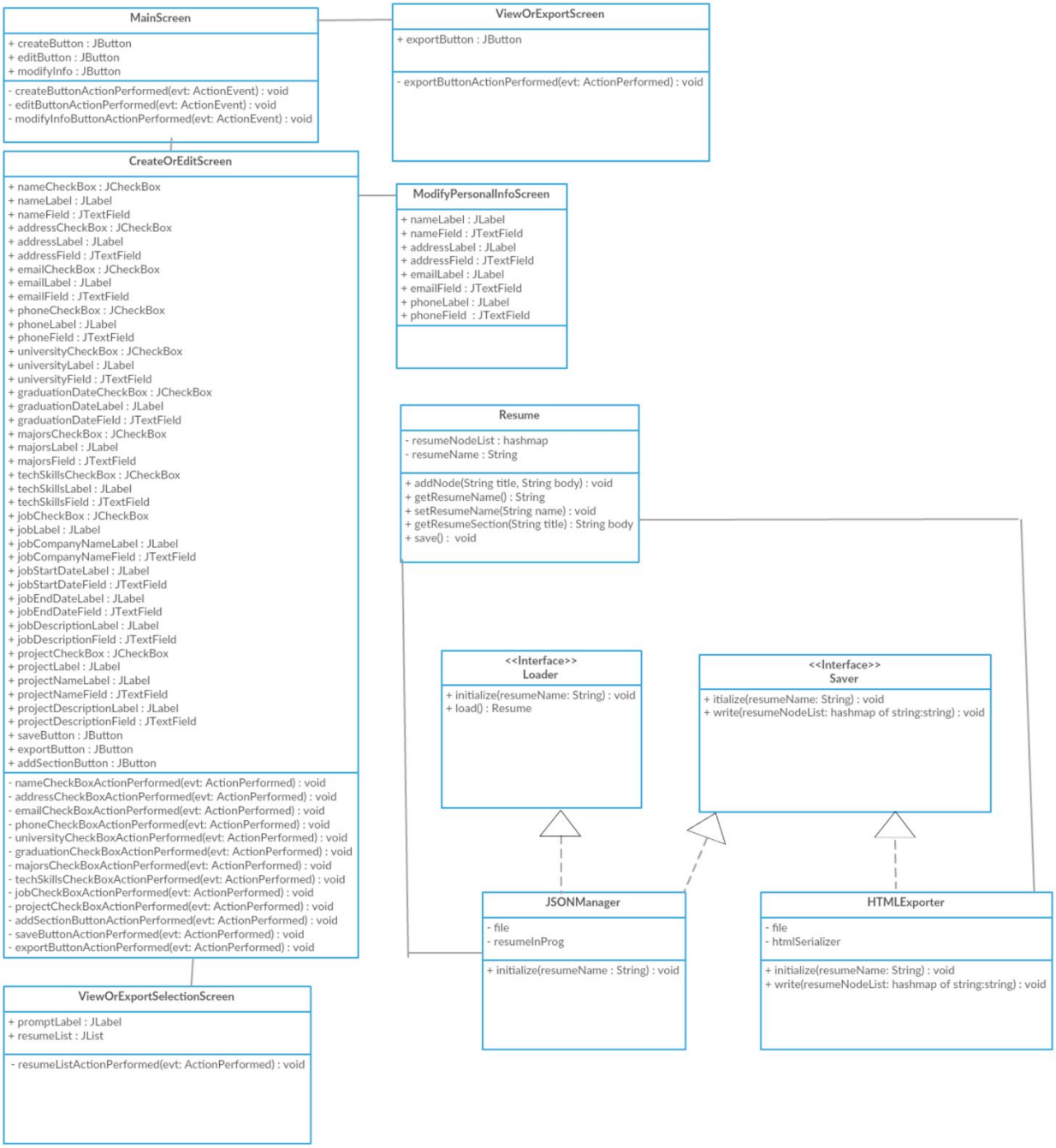
UC-04	User can modify his/her personal information
UC-05	Resumes can be generated and viewed once they've been created
UC-06	Users can add custom fields to their resume e.g. volunteer experience

2. List the features were not implemented from Part 2 (table with ID and title).

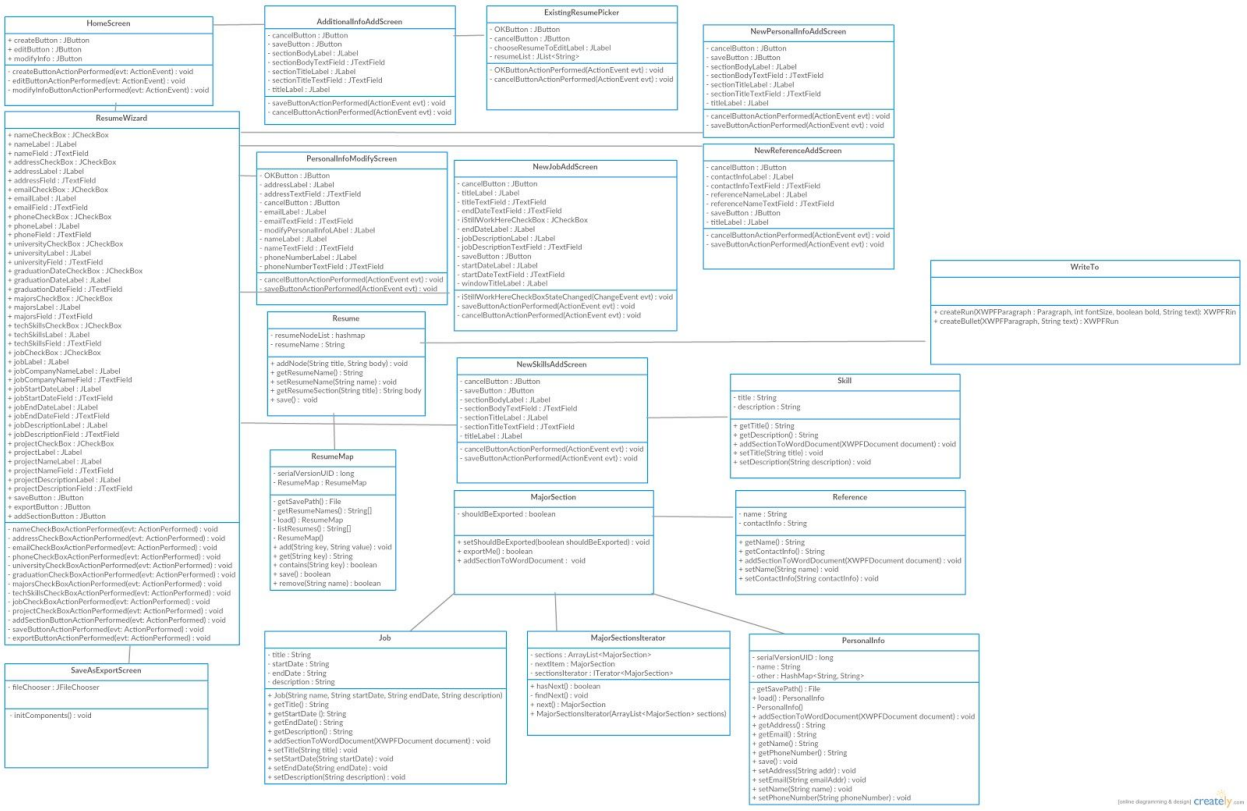
NFR-002	Supporting new file extensions cannot be supported without somewhat major changes to the code

3. Show your Part 2 class diagram and your final class diagram. What changed? Why? If it did not change much, then discuss how doing the design up front helped in the development.

Original class diagram:



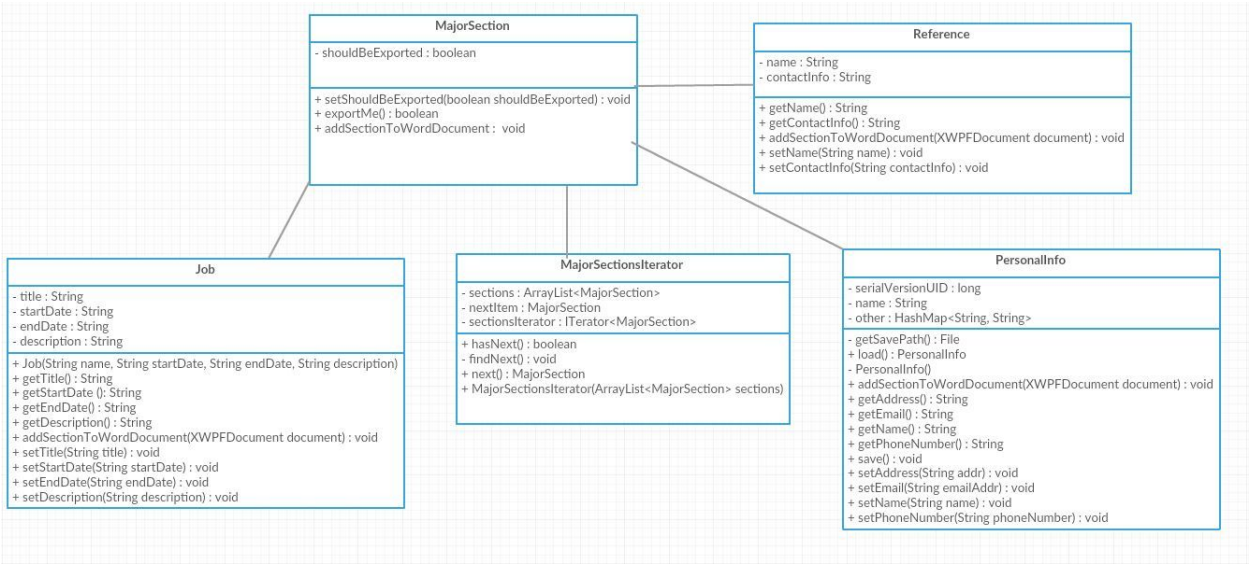
Final Class Diagram:



Quite a bit changed from our original class diagram to the final class diagram shown above. The most substantial change had to do with how we decided to go about saving the resume. We diverged from our initial strategy in order to store resumes using a Resume object and accompanying objects such as job, reference, skills, etc. In addition we needed to make changes to the user interface, which changed the class diagram as well. Those changes we expected to happen.

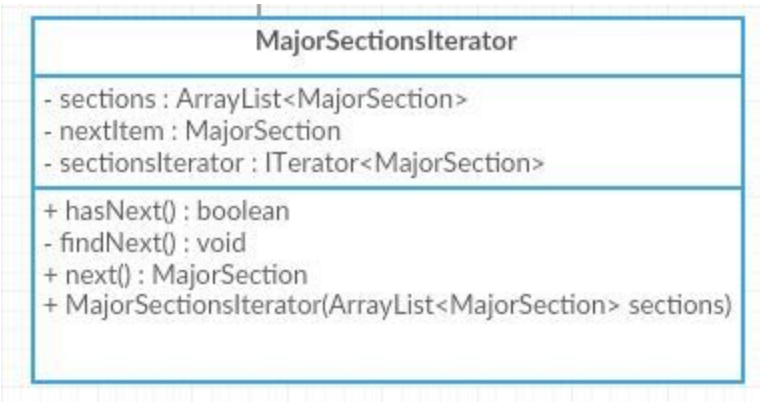
4. Did you make use of any design patterns in the implementation of your final prototype? If so, how? Show the classes from your class diagram that implement each design pattern (each design pattern as a separate image in the .PDF). If not, where could you make use of design patterns in your system? Show a class diagram of how you could implement each design pattern and compare how it would change from your current class diagram.

- Object serialization



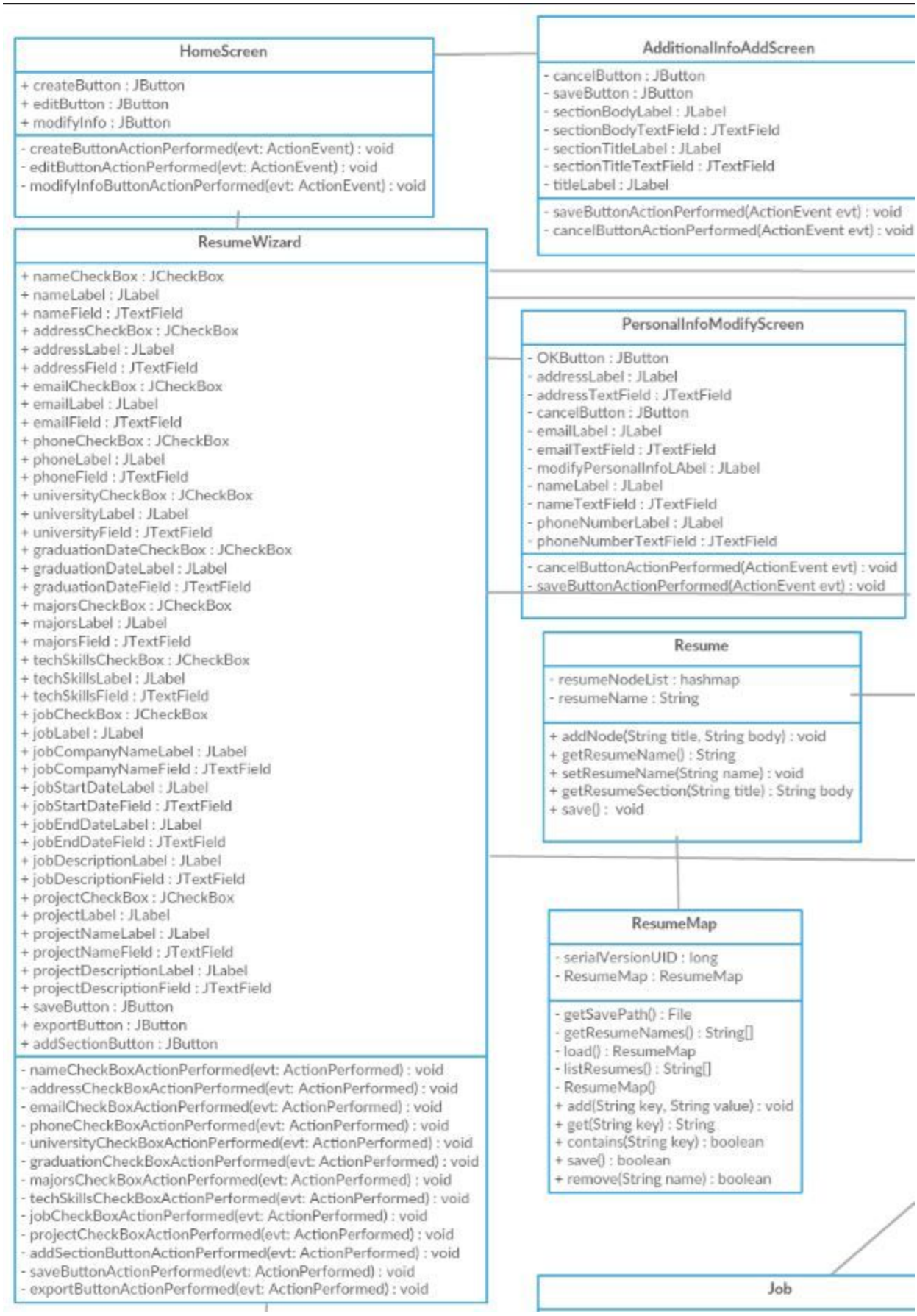
In the MajorSection class we used ObjectSerialization in order to ensure that the values for Job, Reference, and PersonalInfo objects would persist beyond the current Java process.

- Iterator



We used the class MajorSectionsIterator in order to use an iterator to walk through the various sections we wanted to save to a resume. This allowed us to easily traverse the various MajorSections without worrying about how to access the data within the objects.

- Creator pattern: who creates a given object? Specific GUI screen classes were responsible for creating new objects and handing them off to the next screen in situations where that made sense.
    - B closely uses A – B has the initializing data for A


HomeScreen
+ createButton : JButton
+ editButton : JButton
+ modifyInfo : JButton
- createButtonActionPerformed(evt: ActionEvent) : void
- editButtonActionPerformed(evt: ActionEvent) : void
- modifyInfoButtonActionPerformed(evt: ActionEvent) : void
AdditionalInfoAddScreen
- cancelButton : JButton
- saveButton : JButton
- sectionBodyLabel : JLabel
- sectionBodyTextField : JTextField
- sectionTitleLabel : JLabel
- sectionTitleTextField : JTextField
- titleLabel : JLabel
- saveButtonActionPerformed(ActionEvent evt) : void
- cancelButtonActionPerformed(ActionEvent evt) : void
ResumeWizard
+ nameCheckBox : JCheckBox
+ nameLabel : JLabel
+ nameField : JTextField
+ addressCheckBox : JCheckBox
+ addressLabel : JLabel
+ addressField : JTextField
+ emailCheckBox : JCheckBox
+ emailLabel : JLabel
+ emailField : JTextField
+ phoneCheckBox : JCheckBox
+ phoneLabel : JLabel
+ phoneField : JTextField
+ universityCheckBox : JCheckBox
+ universityLabel : JLabel
+ universityField : JTextField
+ graduationDateCheckBox : JCheckBox
+ graduationDateLabel : JLabel
+ graduationDateField : JTextField
+ majorsCheckBox : JCheckBox
+ majorsLabel : JLabel
+ majorsField : JTextField
+ techSkillsCheckBox : JCheckBox
+ techSkillsLabel : JLabel
+ techSkillsField : JTextField
+ jobCheckBox : JCheckBox
+ jobLabel : JLabel
+ jobCompanyNameLabel : JLabel
+ jobCompanyNameField : JTextField
+ jobStartDateLabel : JLabel
+ jobStartDateField : JTextField
+ jobEndDateLabel : JLabel
+ jobEndDateField : JTextField
+ jobDescriptionLabel : JLabel
+ jobDescriptionField : JTextField
+ projectCheckBox : JCheckBox
+ projectLabel : JLabel
+ projectNameLabel : JLabel
+ projectNameField : JTextField
+ projectDescriptionLabel : JLabel
+ projectDescriptionField : JTextField
+ saveButton : JButton
+ exportButton : JButton
+ addSectionButton : JButton
- nameCheckBoxActionPerformed(evt: ActionPerformed) : void
- addressCheckBoxActionPerformed(evt: ActionPerformed) : void
- emailCheckBoxActionPerformed(evt: ActionPerformed) : void
- phoneCheckBoxActionPerformed(evt: ActionPerformed) : void
- universityCheckBoxActionPerformed(evt: ActionPerformed) : void
- graduationCheckBoxActionPerformed(evt: ActionPerformed) : void
- majorsCheckBoxActionPerformed(evt: ActionPerformed) : void
- techSkillsCheckBoxActionPerformed(evt: ActionPerformed) : void
- jobCheckBoxActionPerformed(evt: ActionPerformed) : void
- projectCheckBoxActionPerformed(evt: ActionPerformed) : void
- addSectionButtonActionPerformed(evt: ActionPerformed) : void
- saveButtonActionPerformed(evt: ActionPerformed) : void
- exportButtonActionPerformed(evt: ActionPerformed) : void
PersonalInfoModifyScreen
- OKButton : JButton
- addressLabel : JLabel
- addressTextField : JTextField
- cancelButton : JButton
- emailLabel : JLabel
- emailTextField : JTextField
- modifyPersonalInfoLAbel : JLabel
- nameLabel : JLabel
- nameTextField : JTextField
- phoneNumberLabel : JLabel
- phoneNumberTextField : JTextField
- cancelButtonActionPerformed(ActionEvent evt) : void
- saveButtonActionPerformed(ActionEvent evt) : void
Resume
- resumeNodeList : hashmap
- resumeName : String
+ addNode(String title, String body) : void
+ getResumeName() : String
+ setResumeName(String name) : void
+ getResumeSection(String title) : String body
+ save() : void
ResumeMap
- serialVersionUID : long
- ResumeMap : ResumeMap
- getSavePath() : File
- getResumeNames() : String[]
- load() : ResumeMap
- listResumes() : String[]
- ResumeMap()
+ add(String key, String value) : void
+ get(String key) : String
+ contains(String key) : boolean
+ save() : boolean
+ remove(String name) : boolean
Job

We used the creator pattern in order to decide which classes should be responsible for creating other classes. For example, HomeScreen creates a Resume object because it is able to provide the initializing data for the Resume Object. Another example is AdditionalInfoAddScreen. When the user wants to add a seciton to their resume, ResumeWizard will pass AdditionalInfoAddScreen the instance of Resume so that it can properly modify the Resume object and the changes will be saved.

5. What have you learned about the process of analysis and design now that you have stepped through the process to create, design and implement a system?

We have learned that it is very important to have clear communication throughout every step of analysis and design when working on a group project. We also learned the importance of using the correct tools for each step. Github was an extremely valuable resource for version control and Eclipse was useful for conveniently adding packages like a Word document exporter. Additionally, it is very useful to create design patterns before coding begins because it gives a stylistic guide on how to write the code and keep everyone on the same page. Having the class diagrams as a guide on where to begin coding is a difficult task, but it makes the first steps of programming much easier. We also learned that the plans we made in the beginning changed a fair amount by the end of the project. This helped us learn that even if you have a good design laid out in the beginning, some change is inevitable as you get a better handle on the realities of implementing everything. In addition, we learned that if you focus on maintaining scalable, well designed code from the beginning, you can make your life much easier when the project grows larger than you might’ve expected.