Name: Xinran Wang

1. How did you design your application? That is, how did you choose what classes to create, and what fields and methods each of them would have?

Application designed was performed with the following intents:

- Satisfy all requirements
- Provide a reasonable amount of flexibility for future extensions (not realistic for this assignment but more realistic for a real world project)
- Each class should operate on a similar set of data. For example, the AcademyAwards class shouldn't be responsible to directly accessing/changing data for a Nominee object. If necessary, this should be performed through a function in the Nominee class.
- User interface functionality to be separated from data storage and access.

Design started with determining the most basic unit of information storage, the Nominee class; each line in the data file would be converted to a Nominee object. From there, I decided to distinguish nominees based on whether they are describing a person or a film. This was done because when searching for nominees, films are searched by year while actors are search by name. Originally, the actorsDatabase and filmsDatabase variables were designed as separate classes, to act as an abstraction layer over how the data is stored (Mappings of name/year to Sets of Nominee objects). The AcademyAwardsSearch class essentially acts as a front-end to all the data. While working on this step, I decided to wrap all the data in the AcademyAwards object, so that this can act as an abstraction layer over the data access. Any potential user interfaces, whether through command prompt or a GUI.

At this point, the two _Database classes had a lot of overlapping methods so theoretically an interface/abstract database class could be designed from which these two could inherit from. However, because this project only required two such types of data storage, I decided to skip designing the interface. Instead, the actorsDatabase and filmsDatabase classes were then removed and refactored into the AcademyAwards class as both a convenient and simple way to organize the code. actorsDatabase and filmsDatabase were both set up as similar data structures since they had similar purposes of storing Nominee data. The key type for the maps were chosen to provide the easiest way to sort and access the data. On a real project, a quick database could be set up to with a Nominees table, to provide an easy way to query the data.

Finally, while designing AcademyAwardsSearch, the NomineeParser library was created to help parse lines in the data file into information for the Nominee class. I decided to separate this code from the Nominee class so that as a result, the Nominee class only acts upon specific Nominee objects whereas NomineeParser operates upon Strings.

AcademyAwardsSearch contains main() method that run the program, and a series of other help methods for String input manipulation, data file and log file access, and AcademyAwards connections. Wrapper methods were written for parsing and reading input from the command line. A method for writing to the log file was also written since it would reused.

2. Describe three specific and distinct ways in which the design of your classes is "good".

1. Separation of classes so that the methods in each class operate upon similar data. For example, AcademyAwards does not try to change or set up Nominee objects. This was done to try to minimize coupling (though there really aren't that many different classes I guess).
2. Access to methods and variables are as restrictive as possible. Only the necessary methods are exposed to the outside.
3. Data is abstracted to hide implementation details.
   Example:
   - Nominee abstracts the categorization of actor vs film
   - AcademyAwards abstracts how actorsDatabase and filmsDatabase is implemented. Someone working with AcademyAwards can search for actors and films but does not need to know that they are implemented as Maps.

3. Describe three specific and distinct ways (not related to design) in which the style of your code is "good" in terms of its appearance, identifier names, ease of reading/understanding, etc.

1. Methods and variables have descriptive names that reflect their functionality.
   Examples:
   a. `parseStringIntoNominee`
   b. `searchForBestPictureWinnerByYear`
2. Variables are initialized close to where they are used to minimize average span.
3. The code follows proper indentation and naming rules consistent with general "good practice" patterns.

4. How do you know your code works correctly? Describe in detail the steps you took to ensure the correctness of your implementation.

Smaller functions were checked using unit tests (see NomineeTest, NomineeParserTest, AcademyAwardsTest). Once those were verified. The overall program (AcademyAwardsSearch) was tested by actually running through the program with comprehensive user inputs. Improper data files, log files, and file accessibility were all tested with dummy files.