# Final Project

This project is worth 15% of your final mark and consolidates all of the things you've learned this term. It also gives you the option of learning about something new that wasn't covered in the course.

## Project Specs

Write a program that uses a GUI to allow the user to view records in a file. You decide what you want to keep track of (media such as book/movie/cd collection, inventory in a store, members of a group or organization, etc). Your program must include the following:

* A file that contains records used in your program (you can use a sequential file or a random access file).
  + Use a sequential file if you are going to be adding records and read them back sequentially. You can also use a sequential file to perform searches that will need to iterate through the entire file.
  + Use a random access file if you are going to be adding and editing records, or going directly to specific records (for example, by allowing a user to select a record by clicking on something in a list box).
  + Deleting records is cumbersome in both kinds of files. However, if you're allowing the user to delete, then obviously you're also allowing edits. In this case, you'd want to use a random access file.
* A class that models a record in your file. For example, if your file contains book information with the fields ISBN, Title, and Author, then you will need a Book class with data members for ISBN, Title, and Author (and other appropriate code and members such as constructors, accessor/mutator methods, toString, etc).
* A class that models the list of records, using an ArrayList (or other Collections class) of record objects (e.g. I might have BookList class of Book objects). This class will perform operations such as adding Book objects to the list, removing book objects from the list, replacing modified book objects in the list, searching for specific book objects, reading in Book objects from a file, saving the list of book objects to the file, etc. The list acts as the data file while the program is running: load the list with book objects at program startup, then on exit, overwrite the file with the book objects from the list. Don't perform add/edit/delete operations on the file directly as they occur - use the list.
* A class that contains the GUI for your program.

Your application must include, AT MINIMUM, the following functionality:

* The ability to ADD, EDIT, and DELETE records in the file.
  + Your user should be able to add, edit, and delete records, in a professional and user-friendly manner.
  + This must also include things like proper data validation.
  + If the user is in the middle of adding or editing a record, they should have the option to cancel the operation and go back to the record they were viewing before they started the add/edit operation.
  + The user should be prompted for confirmation when deleting a record.
  + All changes to the records should be applied to the data file.
* The ability to NAVIGATE records in the file.
  + For example, you could use a set of First/Previous/Next/Last buttons to navigate the records (the user should not be able to navigate beyond the first and last record).
  + For example, you could populate a list box with records (show only 1 or 2 fields as appropriate) and the user can select a record from the list to view the rest of the fields.
* The ability to SEARCH for records in the file.
  + The user should be able to enter a search key and then be shown a list of records that match that search key (e.g. a search key of "fred" on a first name field would yield records with the first name "Fred", "Frederick", or "Alfred").
  + The user can choose one of the matching records and view that record's details on the main screen.
  + You only need one search so you can choose what field you'd like to make available for searching (for example, if your application kept track of your video game collection, you might wish to search by title.
  + If a search yields no result, an appropriate message should be displayed.

If you'd like to see a demonstration of a sample final project, please ask your professor. You can also read about and examine screen shots of a basic example application that maintains a list of Sheridan courses.

We will be doing various small assignments and exercises in class that demonstrate many of the project components. It will be up to you to assemble these parts into a functional application.

### Extra Functionality for Bonus/Teams

If you are striving for extra bonus marks, or working with a partner and are looking for ways to make the project more appropriate for two people, here are a few ideas:

* Allow filtering using combo boxes. For example, if you're choosing to show your records in a list box for navigation, you can add a combo box to filter the list. For example, if I were keeping track of video games, I might list the game titles in the list box, and then have a combo box that listed the different gaming consoles. Then the user could filter the list of games by selecting a specific gaming console. Don't forget to include an option in the combo box so that the user can view all of the records again!
* Sort your record objects in some order. For example, continuing with the above example, you could sort the games in the list by game title.
* Add a screen that shows one or more "reports" or summaries of your data. For example, perhaps you'd like to see a list of your most expensive video games (user could enter the price range), or maybe you'd like to see a list of games grouped by genre, with the # of games in each genre.
* Add an extra file that is used as a "look up". For example, if you already have a games file and one of your fields is for the type of gaming console, it's standard practice to store the gaming console names with ID values in a separate file. The console field in the main games file will record the ID number of the console. Then when viewing/adding records, you can load a combo box with the console names from the console file, and when the user chooses a console, you store the corresponding ID in the games console field. This eliminates user error when entering the game console name while adding/editing records - they can just pick one from a combo box!
* The above can also be done with an enumeration ("enum"). This is an advanced topic but is also standard practice in industry. It's not very hard to learn if you already understand constants (final variables) and OOP. Ask your professor if you'd like more information and an example of an ENUM.

## Proposal: Having your Project Approved

**You must submit a project proposal in order to receive any marks for your final project submission. Projects submitted without an on-time proposal will be given a grade of 0.**

**Late proposals: Late proposals will result in a 20% penalty on the overall final project grade. Late proposals will not be accepted after July 31st; if you don't submit a proposal by this time then you receive a 0 for your final project.**

The proposal must contain the following information:

1. Your first page (can be a title page or simply the first page of your proposal) must include the following:
   * Your full name and student number (if you have a partner, be sure to include their name & student number, also.
   * The title of your proposed application (e.g. DVD Inventory System)
2. A description of your proposed project in proper paragraph form with correct spelling and grammar. This paragraph should answer the following questions:
   * What kind of data are you keeping track of?
   * Who is the project intended for; who is the user?
   * Why is this a useful application? What purpose does this application serve?
   * What are the main features and functions of your application?
   * Anything else you feel you need to say in order to describe your proposed application.
3. A file description that list the description and data type of each field.
   * This must be done in proper chart form as shown in the example below.

Example: the chart below describes the format of records in a file that tracks information about bottles of wine:

|  |  |  |
| --- | --- | --- |
| **File: wine.dat** | | |
| **Description** | **Data Type** | **Comments** |
| wine i.d. | String | uniquely identifies a bottle of wine |
| estate | String | the name of the place that made the wine |
| grape | String | the type of grape e.g. chardonnay |
| year | int | the year the wine was made |
| price | double | the wine's current value |
| qty | int | the number of bottles I own |

1. Screen designs or sketches, either done by hand or with a tool such as Visio, Microsft Word, or other graphics/charting software
   * Your screen designs might change as you develop your program, but they shouldn't change drastically.
2. A UML diagram of the classes used in your application. If you haven't learned UML yet, you can see lots of examples in your textbook.
3. A description of any introductory research you might need to do about any additional classes/packages or concepts you might need to learn about in order to develop your program.

Submit your proposal in the appropriate DropBox in SLATE at the date/time specified by your professor. **Emailed submissions will not be accepted.** Late submissions will only be accepted for up to 7 days will be penalized. **Submissions more than 7 days late will not be accepted and will result in a project grade of 0**.

## Evaluation

Evaluation of this assignment is based on:

* Interface design - neat, professional, user-friendly, easy to use and not confusing or ambiguous, gives good feedback (e.g. error messages, instructions).
* Program functionality - does it work properly? Does anything make it crash? Does it meet the program requirements?
* Quality of Code - code is easy to read and interpret, good spacing and indentation, proper and complete documentation, efficient use of program control structures, appropriate method and variable names, efficient code that is not redundant or unstructured, use of constants instead of literals where appropriate, etc.

The largest part of the evaluation is the program's functionality. The program should not crash under any circumstances or an automatic grade of 0 is assigned. This means you need to thoroughly debug your program and ensure that your interface is mistake-proof! Your program should also include professional and user-friendly feedback (e.g. instructions, error messages) and there should be no ambiguous elements to your program. If necessary, you may be asked to demonstrate your program to the instructor.

### Documentation:

Each class/program must be fully documented, including programmer identification.

You must include proper javadoc comments for all classes in your project, although they are not required for your GUI class. So any libraries you've written and the classes you wrote that aren't GUI classes should all include correct javadoc documentation.

You DO NOT need to compile and include the javadoc files in your project. Just make sure you have the complete and correctly formatted javadoc comments in your code.

## Submission

You must ZIP or RAR all of the required files (see below) into one submission file. ***Read this carefully! If you do not follow the instructions properly, you will lose marks***.

Your submission file must include:

1. **Your Netbeans project directory.** As usual, just zip/rar your entire project directory into one file.

This file should have a meaningful name for your project with "SOURCE" or "SRC" in it. For example, **"BibsBookTracker\_SRC.zip"**.

Upload your submission files to the appropriate SLATE DropBox.