# CSCI 3901 Final Project

Due date: 4pm Tuesday, December 14, 2021 in git.cs.dal.ca. The repository is at https://git.cs.dal.ca/courses/2021-fall/csci3901/project/xxx.git where "xxx" should be replaced by your timberlea login id. The repository is already created for you, so you should just need to clone it.

## Goal

The course project is your opportunity to demonstrate all of the concepts from the course in one body of work.

## Project Structure

The final project will have you apply your problem solving and development skills. The solution will require you to bring together

- Abstract data types and data structures
- Java implementation
- Basic algorithms
- Software development techniques including version control, testing, debugging, and defensive programming
- Good software program design
- Database design and use

While we provide a recommended problem for the course, you have the option of proposing a different problem for the project. A project proposal must include a non-trivial example of all the concepts mentioned above.

The project milestones do not change if you choose to propose your own problem.

This project is _not_ expected to include a user interface component or software that directly accesses a device's hardware.

## Recommended Problem

The prevalence of digital pictures, especially from smart phones, has increased the number of images and videos of people. At the same time, the abundance of storage space tends to have people not delete pictures or videos that they take. Consequently, finding images and videos increases in difficulty.

Genealogy (the study and tracing of lines of descendants) is well-established for tracing and tracking family relations, but doesn't have good ties into the richer sets of pictures that exist today or to archived pictures that are now being digitized. Ideally, when a genealogist asks about the great grandmother of person X, they would like to get the set of pictures for the individual as well as their name.

This project will create a system that links family tree information with an archive of pictures and the metadata of the pictures.

## *Simplification acknowledgement*

For simplicity, we will only deal with biological family relations. I appreciate and acknowledge that familial relations are much more complex than merely biological. Focusing only on the biological family relations is merely a starting simplification on which more complex familial relations can build in future versions of the code.

Also, the description of familial relations in this project will take a primarily English genealogy perspective in its definitions. Again, I acknowledge that the concept of "cousinship" and other relations is strongly connected to culture. I choose the English genealogy definitions because they have precise definitions. If we can get these definitions working then future versions of the code can include richer relation definitions.

## *The project*

The two major components of the project are the family tree database and a media archive.

The family tree database will store information about individuals and will store relations between individuals. Information that we store about individuals will include

- Their name
- Date and location of birth
- Date and location of death
- Gender
- Occupation
- References to source material
- Notes on the individual

The relations that we store in the tree include

- Parent/child relations
- Partnering ceremony relations (eg. marriage)
- Partnering dissolutions (eg. divorce)

The media archive manages metadata of different forms of media. We describe "media" as pictures, but it could be other forms of media like videos. For any picture in the archive, it includes

- The filename of the picture on some central hard drive (so we aren't storing the pictures directly in the archive)
- The date when the picture was taken
- The location of the picture, possibly as
  - A location name like "Dalhousie University"
  - A city name, province, or country name

- Tags for the picture, which could identify a trip name of the picture or some other tags
- The individuals seen in the picture

Given the nature of the material, any single individual in the family tree and any picture may only have a subset of the information. What we are guaranteed is that

- any individual in the family tree has a name and
- any picture in the media archive has a filename.

Much of the other information may be absent. Also, dates may be partial, like just a year or a year and a month (but never a day alone or a month and day alone).

With information in hand, we will want to help genealogists answer question like

- how are persons X and Y related
- show me the references and notes for person X
- list the descendents of person X for Z generations
- list the ancestors of person X for Z generations
- list the pictures of a particular tag within some time range
- list the pictures of a particular place within some time range
- list the pictures of a given set of people (like a couple) within some time range chronologically
- list the pictures that include all the immediate family members of person X (immediate children)

Your task is to create the classes that will gather the information for this system, store it in a database, and resolve these queries.

## *Defining Relations*

Using a biological tree simplifies how we define relations between two individuals. Relations are defined by two parameters: degree of cousinship and levels of separation.

Consider a family tree where your ancestors are above you in the tree and your descendants are below you in the tree. Given two individuals, X and Y, find their nearest common ancestor in the tree, who I will call Z. We can then count how many generations separates each of X and Y from Z. Suppose that Z is 4 generations back from X and that Y is 2 generations back from Z (so X is the great great grandchild of Z and Y is the grandchild of Z). The smaller of the generations back (2 in this case) minus 1 defines the cousinship between X and Y and the difference in the generations (4-2 = 2 in this case) represents the levels of separation. In this example, X and Y are then first cousins twice removed.

More generally, if Z is nX generations back from X and nY generations back from Y then X and Y are (min{nX, nY} -1) cousins |nX – nY| removed. When one of nX or nY is zero then we have an ancestor/descendent relations. When min{nX, nY} = 1 we have special names of sibling, aunt, uncle, niece, or nephew (or the gender-neutral terms of siblings, pibling, and nibling) and we express the level of separation with “grand” and “great” prefixes.

Consider the biological family tree in Figure 1. Given these rules, we then get the following relations

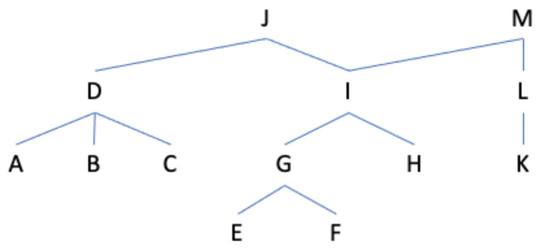

*Figure 1: Sample family tree*

| Person 1 | Person 2 | Nearest common ancestor | Degree of cousinship | Degree of removal | Relation name |
|---|---|---|---|---|---|
| A | B | D | 0 | 0 | Sibling |
| A | E | J | 1 | 1 | First cousins once removed |
| E | J | J | -1 | 3 | Ancestor/Descendent relation – so E is the great grandchild of J and J is the great grandparent of E |
| F | H | I | 0 | 1 | Sibling once removed – so F is the nibling of H and H is the pibling of F |
| C | K | None | None | None | Not biologically related |

This project will ask you to report relations in terms of cousinship and degree of removal. It will not ask you to translate that into a textual relation name.

## *Added classes*

The functionality is defined based on two classes, which you must create:

- PersonIdentity – an identifier for an individual in the family tree
- FileIdentifier – an identifier for the file in the media archive
- BiologicalRelation – a class that specifies the cousinship and the level of removal in a relation

## *Manage the family tree*

PersonIdentity addPerson( String name )

Add an individual to the family tree database.

Boolean recordAttributes( PersonIdentity person, Map<String, String> attributes )

Record information about an individual in the family tree database. Each attribute name is the key to the Map. Examples of attributes are “date of birth”, “gender”, and “occupation”. The Map stores the attributes as Strings, but you are allowed to store them in different formats to make other operations easier to perform.

Return true if all attributes were stored.

Boolean recordReference( PersonIdentity person, String reference )
Record a source reference material for the individual. A person can have multiple source references for them. Examples of a reference could be the location of a birth certificate or a web page that lists when someone graduated.

Return true if the reference was stored in the system.

Boolean recordNote( PersonIdentity person, String note )
Record a note for the individual. A person can have multiple notes for them.

Return true if the note was stored in the system.

Boolean recordChild( PersonIdentity parent, PersonIdentity child )
Record a parent/child relation for the individuals.

Return true if the relation was stored in the system.

Boolean recordPartnering( PersonIdentity partner1, PersonIdentity partner2 )
Record a symmetric partnering relation between the individuals.

Return true if the relation was stored in the system.

Boolean recordDissolution( PersonIdentity partner1, PersonIdentity partner2 )
Record a symmetric dissolution of a partnering relation between the individuals.

Return true if the dissolution was stored in the system.

*Manage the media archive*
FileIdentifier addMediaFile( String fileLocation )
Add a media file to the media archive. Return a media file identifier, which will be used to represent the file in the rest of the archive functions.

Boolean recordMediaAttributes( FileIdentifier fileIdentifier, Map<String, String> attributes )
Record information about a media file in the archive. Each attribute name is the key to the Map. Examples of attributes are “year”, “date”, and “city”. The Map stores the attributes as Strings, but you are allowed to store them in different formats to make other operations easier to perform.

Return true if all attributes were stored.

Boolean peopleInMedia( FileIdentifier fileIdentifier, List<PersonIdentity> people )
Record that a set of people appear in the given media file.

Return true if the people are now connected to the medial file in the system.

Boolean tagMedia( FileIdentifier, fileIdentifier, String tag )
Record a tac for a media file. A media file can have many tags.

Return true if the tag was stored in the system.

*Reporting*

PersonIdentity findPerson( String name )
Locate an individual in the family tree.

FileIdentifier findMediaFile( String name )
Locate an individual in the family tree.

String findName( PersonIdentity id )
Return the name of an individual.

String findMediaFile( FileIdentifier fileId )
Return the file name of the media file associated with the file identifier.

BiologicalRelation findRelation( PersonIdentity person1, PesonIdentity person2 )
Report how two individuals are related.

Set<PersonIdentity> descendents( PersonIdentity person, Integer generations )
Report all descendents in the family tree who are within "generations" generations of the person. Consider the children of "person" as being 1 generation away.

Set<PersonIdentity> ancestores( PersonIdentity person, Integer generations )
Report all ancestors in the family tree who are within "generations" generations of the person. Consider the children of "person" as being 1 generation away.

List<String> notesAndReferences( PersonIdentity person )
Return all the notes and references on the individual, returned in the same order in which they were added to the family tree.

Set<FileIdentifier> findMediaByTag( String tag , String startDate, String endDate)
Return the set of media files linked to the given tag whose dates fall within the date range. Null values for the dates indicate no restrictions on the dates.

Set<FileIdentifier> findMediaByLocation( String location, String startDate, String endDate)

Return the set of media files linked to the given location whose dates fall within the date range. Null values for the dates indicate no restrictions on the dates.

For simplicity, we require that the location name be a perfect match. For example, a query location of "Halifax" will not match a stored location of "Halifax, Nova Scotia". You can go beyond this limitation if you want.

List<FileIdentifier> findIndividualsMedia( Set<PersonIdentity> people, String startDate, String endDate)
Return the set of media files that include any of individuals given in the list of people whose dates fall within the date range. Null values for the dates indicate no restrictions on the dates. Return the files in ascending chronological order (breaking ties by the ascending order of the file names). List files with no dates at the end of the list.

List<FileIdentifier> findBiologicalFamilyMedia(PersonIdentity person)
Return the set of media files that include the specified person's immediate children. Return the files in ascending chronological order (breaking ties by the ascending order of the file names). List files with no dates at the end of the list.

## *Milestones*

Only the final submission is graded. The intermediate milestones appear as points to get feedback on the project:

- November 5: High-level breakdown analysis of the problem
- November 12: Blackbox tests and plan of feature development
- November 17: External documentation of data structures, code design, key algorithms, and any additional white box tests
- November 30: Implementation report in relation to earlier plan; git submission showing some progress on implementation
- December 14: Project due

All milestone material is due in the course git repository.

## *Marking scheme*

- Meeting intermediate milestones (10%)
- Documentation (10%)
- Overall design and coding style (15%)
- Working implementation, including efficient processing (50%)

| | |
|---|---|
| Family tree management | 10% |
| Media archive management | 10% |
| Reporting | 25% |
| Efficiency | 5% |

- Test plan (15%)