CSCI 3901 FINAL PROJECT – EXTERNAL DOCUMENTATION

Instructor: Dr. Michael McAllister
Name: Nadipineni Hemanth Kumar
Banner ID: B00899473
Date: 2021-12-13

Overview:

The overview of this project is to develop a system that connects family tree information with a photo repository and the metadata associated with the photos. The system consists of a family tree database and a media archive. The system will store the information of individuals and their relations in the family tree database. The system will store the information of photos in the media archive. With all this information, we need to give output for the queries given such as a list of ancestors of person X for Z generations.

Files and External Data:

The program will contain three main classes: PersonIdentity class, FileIdentifier class, BiologicalRelation class. The program will also have a Genealogy.java class to manage the family tree, media archive, and reporting. The program has helper classes such as MySQL classes inside the database package to help with database connection and database queries. Also, validator classes inside validator package to validate input data and check the existence of person/media inside the family tree database. On top of that, the program consists of populator classes to populate the fileidentifier, personidentifier, biologicalrelations, familytrees from the database data whenever the program started. So that, the program will automatically create the tree during the start and keep adding to the same tree.

Project Structure:

- Genealogy class: This is the class with all the methods specified in the project document. This acts as an overarching class to pull all the methods together. The purpose of this class creation is to achieve the overarching in such a way to interact with the methods listed in the project statement.

- Main class: This is the main class that creates a genealogy object and calls all the methods to pass the parameters into the methods from genealogy. The purpose to create this class is to have a general class to have interactions with the user and to pass the arguments.

- /classes:
    - BiologicalRelation class: This class is used to create objects with the data related to a person's relations and other important data such as degree of cousinship and degree of removal. The purpose of this class creation is to facilitate the benefit of storing relations into an object.
    - PersonIdentity class: This class contains all the data attributes, ID and name of a person. This is being used to create a person identity object with all the data related to a person. This file was created to have the benefit of storing all the data related to a person into the objects.
    - FileIdentifier class: This class contains all the attributes, ID and fileLocation of a media file. This is being used to create media objects with all the data related to a media. This file was created to have the benefit of storing all the data related to a media file into the objects.
    - FamilyTree class: This class contains all the data related to the family tree such as parent/child, partners, dissolutions. The purpose of this class creation is to facilitate the benefit of storing the family members and other relations.

- /database:
    - MySQLConnector class: This class is used to make connections to the database specified. This class is created to have the required database related data and to initialize a MySQL connection.
    - MySQLQuery class: This class is used as a MySQL querier. This class has the methods to run queries to store data passed from the genealogy class. This class is created to have the queries in one place.
    - MySQLPersonAttributes class: This class is to run MySQL queries to store data related to a person. This class is created to have all the queries related to a person in one place
    - MySQLMediaAttributes class: This class is to run MySQL queries to store data related to a file. This class is created to have all the queries related to a media file in one place.

    - MySQLRelations class: This class is to run MySQL queries to store data related to relations. This class is created to have all the queries related to relations in one place.
    - MySQLFetcher class: This class is to run MySQL queries to fetch data from the database. This class is created to have all the queries to fetch data in one place.
- /validators
    - Validator class: This class is used to check validations for the existence of fileIdentifiers, personIdentifiers…etc. in the database. This class is created to have the validations for incoming objects such as person, fileId, at one place.
- /populators
    - GenealogyPopulator: This class is used to populate the fileIdentifier, personIdentity, biological relation objects from the data collected from the existing database and creates a family tree at the start of the program. This class is created to have an auto-populatoer for objects.

/test

- /assignment
    - GenealogyTest class: This class is a Junit test class used to test the functionality of the Genealogy class by passing various types of attributes. The purpose of this class creation is to have the test cases for Genealogy

Data Structures and Their relations to each other:

- This program uses Arraylists to store fileIdentifiers and personIdenty objects. (List<PersonIdentity> personIdentityList, List<FileIdentifier> fileIdentifierList)
- This program uses HashMaps to create FamilyTree. It also uses a HashMap to store fileIdentifiers. (Map<Integer, FamilyTree> familyTreeMap, Map<Integer, FileIdentifier> fileIdentifierMap)

The data will be taken from the Main class as input or hard coded for simplification. This data will be passed into addPerson, recordPersonAttributes, addMedia, recordMediaAttributes….etc,. functions. This will create the objects of personIdentity and fileIdentifiers and add them to the respective list and maps. Family Tree is being created using a node kind

of structure but not just with just left node and right node. Using a list for parents, a list for children, and a map for the partners. Using the lists because of the possibility of having more than two children for one person. So, the structure won't' appear as a binary tree. This way, one parent can have more than 2 or 100 children as discussed in the previous milestone.

While finding relations the program will traverse through the map of familytree and find the level of separation to find the generation gap and also find the common ancestor through the traversal of the map.

Key Algorithms:

The algorithms that can be used on this is like Depth-first search and level order traversal where the program starts from the root and goes towards the children. Then find the level at which they are present and by subtraction, this can report the level of separation and from the level of separation and cousinship, the relation of two persons is found from the formulae mentioned in the breakdown analysis. For example, let's there is a person1 and he has three children person2, person3 and person4, and let's say only person4 has one child person5. Now if we want to find the relation or the generations gap between person1 and person5, we can start from the root(person1) and then go down (incrementing levelCounter++) and check if it's person5, if not go down and check if person5 or if it's null go back to the previous level and continue the same with another child. This way we can perform the algorithm. Instead of going down after person2, we can go breadthwise. person1->person2->person3->person4->person5 (found). A depth-first search is being used because every parent can have way more children.

- The Genealogy() constructor will populate the objects from the existing records in the database if any.
- Then the data from the Mian class if any will create more objects and store those objects inside a list and map.
- The same goes with the media data too.
- Then for reporting the data will be fetched from these object maps.

- For reporting relations, traversal through the map of partners, child list and parent list will be done.
- This will report back the degree of cousinship and the degree of removal will be calculated.

  Formulae:
  - Degree of cousinship: min(nA, nB)-1
  - Degree of removal: |nA-nB|

  Where A – a person named ‘A’

  B – a person name ‘B’

  nA - the number of generations A is back from the common ancestor

  nB - the number of generations B is back from the common ancestor

  From the above data, the relation will be reported:
  - Ancestor/Descendent Relation: when either nA or nB is 0.
  - Sibling: when min{nA, nB} is 1.
  - X cousins Y removed: where X = Degree of cousinship, Y = Degree of removal.
  - Sibling Y removed: when min{nA, nB} is one and degree of removal>0; Y = Degree of removal

The methods inside the Genealogy class:

- Genealogy(): This is a constructor being used to repopulate objects from the existing data in the database.
- addPerson(): This method is being used to record the name of the person passed from the main class and also to create a person identity object. This method will also store the name of the person in the database and pushes the personIdentity object into personidentity list and family treemap.
- recordAttributes(): This method is being used to record the attributes of the person related to the personIdentity object passed. This method will also push the data into the database.
  - addAttributes(): This is a helper method for repetitive calls to record person data. This method helps recordAttributes.

- recordReference(): This method is created to store the references in the list linked to personIdentity and also to store them in the database.
- recordNote(): This method is created to store the notes in the list related to personIdentity and also stores it in the database.
- recordPartnering(): This method is created to record the partnerings and store the relations in the database.
- recordDissolution(): This method is created to record the dissolutions of persons and store them in the database.

- addMediaFile(): This method will take the arguments to store the fileLocation in the database and create fileIdentifier object and pass the object to fileIdentifierMap and fileIdentifierList.
- recordMediaAttributes(): This method takes the arguments of attributes related to the media and stores them to the fileIdentifier object and also inside the database.
    - addmediaAttributes(): This method is a helper the recordMediaAttributes() calls the iterative calls.
- tagMedia(): This method takes the arguments of tags for the media and stores them in a tag list related to the fileIdentifier. It also passes tags to the database.
- peopleInMedia(): In this method the arguments such as fileIdentifier and a list of people present in that media. This method will store the persons in the people list related to the file identifier and store these people inside the database.

- findPerson(): This method is created to traverse through the familyTree map and find the personIdentity object having the same name.
- findMediaFile(): This method is created to find the fileIdentifier object having the same fileLocation as same as the arguments passed.
- findName(): This method is created to find the name of the person related to the person Id passed. This method will traverse through the map of objects and find the person.
- findMediaFile(): This method will return the fileLocation for the file identity object passed as an argument.

- findRelation(): This method is created to find the relation between two persons passed as arguments of personIdentity objects.
    - levelOfNode(): This method is created to find the level at which the given person is present.
    - commonAncestor(): This method is created to find the common ancestor of the given persons.
    - checkParentChilds(): This method is created to check the parent and children by adding each traversed person into a list for each person and comparing the common.
- ancestors(): This method is created to find the ancestors of a person being within a generation away. This method uses the familyTree map to travers and compares the generations and the number of traversals.
- descendants(): This method is created to find the descendants of a person being within a generation away. This method uses the familyTree map to travers and compares the generations and the number of traversals.
- notesAndReferences(): This method is created to report any notes and references related to a person passed as an argument. It travels through the family tree map and finds the personIdentity object and retrieves the notes and references list linked to that person and reports it back.
- findMediabyTag(): This method is created to report the fileidentifiers with tags of a media within a given date range. This method will traverse through the fileIdentifierMap and find the file that is passed and reports back the fileIdentifiers linked to that file.
    - tagMediaHelper(): This is a helper method for findMediabyTag() to call iterations and check the file and dates.
    - nullDatesMediaTags(): This is a method linked to findMediaByTag. This method is helpful when any of the dates are passed as null objects. This will handle the null conditions as specified in the project document.
- findMediaByLocation(): This method is created to report the fileIdentifiers with files taken at the given location of a media within a given date range. This method will traverse through the fileIdentifierMap and find the file that is passed and report back the fileIdentifiers after comparing the locations linked to that file.

    - nullDatesMediaLocation(): This method is created to handle the null cases related to the start and end date of the given arguments.
- findIndividualsMedia(): This method is created to find the list of fileidentifiers that contains a set of people present in the media file from the given start and end dates. This method will traverse through the people and compares with the people in the list of people present in the filedentifiers of fileidentifierMap and add the person to the list.
    - individualsInMediaHelper(): This method is created to help the findIndividualsInMedia() with the iterations.
- findBiologicalFamilyMedia(): This method is created to fetch the media related to the immediate children of the person passed as argument. This method will traverse through the familytree and find the person and then finds the immediate child and traverse through the media files and through the list of people linked to the media file and if the child is present there, the methods add the corresponding fileIdentifier to that list.

Steps taken for the improvement of efficiency:

The database was designed in the 3NF. Maps were prioritized over the lists. Also, data is being populated at the start of the program to avoid query costs for every reporting. Also, several methods were optimized to increase efficiency.

Database Design:

The database ER diagram has been uploaded. Please refer to “Final – B00899473-Database_Documentation-CSCI3901-Project.pdf”

Note: I referred to the occupation table example given during the class for creating Notes, References, Tag, Occupations, Locations table: to simplify storing multiple attributes.

Assumption:

The assumptions mentioned in the PDF were considered.

Assuming that the database is available to perform operations

The default date format of the MySQL “yyyy-mm-dd” is considered. The date attributes for personIdentity and media archive can be partial, but not the attributes such as startDate, endDate in reporting functions.

Whenever a partial date is passed to a media file, consider it as 01-01-yyyy or 01-mm-yyyy to make it as a date format.

Consider the following keys:
Person attributes-

```
put("birthDate", "1891-09-19");
put("birthLocation", "Ottawa");
put("deathDate", "1928-19-01");
put("deathLocation", "Toronto");
put("gender", "female");
put("occupation", "engineer");
```

Media attributes-

```
put("date", "2011");
put("location", "Canada");
```

Limitations:

The program is limited only to biological family relations.

It is advisable to mutex to prevent data corruption for simultaneous transactions.

Notes:

Please create tables using the SQL file provided in the repository before testing.

Add JUnit library from Maven and JDBC connector in the repository to the project structure before running

Had a discussion with a friend about the project implementation but the code was never shown or shared.

References:

The project statement PDF.

Lab 6 reports for making database connections.

https://www.softwaretestinghelp.com/black-box-testing/

https://reqtest.com/testing-blog/black-box-testing/

https://strongqa.com/

https://en.wikipedia.org/wiki/Black-box_testing

https://www.guru99.com/white-box-testing.html

https://www.vogella.com/tutorials/JUnit/article.html

https://www.softwaretestinghelp.com/

https://programmingtechie.com/2020/12/26/junit-5-complete-tutorial/#

https://xperti.io/blogs/java-coding-best-practices/

https://medium.com/javarevisited/10-best-practices-to-design-and-implement-a-java-class-489f72a7099a

https://www.w3schools.com/sql/

https://www.geeksforgeeks.org/get-level-node-binary-tree-iterative-approach/?ref=rp

https://www.geeksforgeeks.org/level-order-tree-traversal/

Code Formatting in IntelliJ IDEA