;

Final Project Documentation

Team Members

Meet Nitinbhai Patel

(B00899516)

(*mt631537@dal.ca*)

Subject:

Software Development Concepts

Professor:

Mike McAllister

Overview

People's photographs and videos have risen as a result of the widespread use of digital photography, particularly from smart phones. Simultaneously, people are less likely to erase photos or videos they shoot as they have plenty of storage space. As a result, finding photos and videos becomes more difficult. Genealogy is well-known for finding and monitoring family links, but it lacks strong linkages to today's larger collections of photographs or to previously stored photographs that are now being digitized.

The aim of the project is to build a system that connects family tree information with the archive of pictures and metadata of the pictures.

Files and external data

Program contains the following files:

1. **Main.java**: It is primarily intended to seek input from the user.

2. **Genealogy.java**: This class give easier access to all the methods. It will contain methods which help to store information about individuals and the relation between two individuals. At last, there is a reporting component that reports the information as required. It will answer some of the genealodists questionsso contains the methods that helps user to store information about the media files.

3. **PersonIdentity.java**: This class is used to identify individuals from the family tree. It contains two variable Int personId and String personName which help to create the object of individual. This object helps use to identify the individual from the family tree.

4. **FileIdentifier.java**: This class is used to identify file using fileLocation from media archive (Here we are taking file location instead of file name because name of two file can be same but location of two file is to be different) It contains three variable Int Id, String filename and String fileLocation which help to create the object of each file. This object helps use to identify the file from the media archive.

5. **Biologicalrelation.java**: This class specifies the degree of removal and degree of cousinship in a relation. It contains two variables Int

6. **login.properties:** It contains username, password and database name. Which is then fetched by open() method present in Genealogy class to create the connection with the database.

7. **SQL File**: It contains the queries that are required to create the all the required tables for this project

Design elements and methods

Classes

Genealogy Class

- This java file trace and tracks the family tree relation. This class has three major components.

1. Family Tree Management
    - This component takes information about the individual through various methods and then store it in the database. It is also responsible for storing the relationship between the individual.
2. Media Archive Management
    - This component takes information about the media file through various method am then store it in the database. The media file can a be image or a video.
3. Reporting Functions
    - This component is responsible for reporting the information as required by the user. It will also answer the some of the genealogists' questions.

It is guaranteed that any individual in the family tree and any media file in the media archive have following information

→ Individual in the family tree have name
→ Media in media archive have file name

Methods

Methods that manage family tree database

1. private void addDatabasePerson()
    - This private method is called when the object of the genealogy class is initialized. It basically creates the object of the data which is already present in the database table "personNameInfo".
    - It will create the object that consists of person id and person name and then it will store that object in the map as value. The key of that map would be the person id. This method is placed in the constructor of genealogy class so it will be called automatically.

2. public PersonIdentity addPerson(String name)
    - This method is used to add the individuals in the database
    - It will add person in the table named "personNameInfo" in the database. It will only add the person's name and the id will be assigned automatically.
    - **Input**: Name of person
    - **Return**: PersonIdentity object

3. private boolean checkPersonId(int id)
    - This method basically checks whether the person id exists or not.
    - It will check with the database table "personNameInfo" to arrive on a conclusion whether the person id exists or not.

- **Input**: Person Id
- **Return**: true if id exists else false

4. public boolean recordAttributes(PersonIdentity person, Map<String, String> attributes)
    - It will take person information such as date of birth, gender and Occupation as in the form of map as input. It will also take PersonIdentity object as parameter.
    - This method then store this information in “personAttributeInfo” table
    - **Input**: PersonIdentity Object, Map (Key as column name and value as information provided by user)
    - **Return**: true if data was stored in the database else return false.

5. public boolean recordReference(PersonIdentity person, String reference)
    - This method is used to add references for the individual in the database.
    - The references will be stored in “personReference” table.
    - A person can have multiple references.
    - We are also recording the timestamps when reference is added in the table
    - **Input**: PersonIdentity object, String reference
    - **Return**: true if references was stored in the database else return false

6. public boolean recordNote(PersonIdentity person, String note)
    - This method is used to add notes for the individual in the database.
    - The references will be stored in “personNote” table
    - A person can have multiple notes.
    - We are also recording the timestamps when node is added in the table
    - **Input**: PersonIdentity object, String note
    - **Return**: true if note was stored in the database else return false

7. private int isRecordChildPossible (int parent, int child)
    - his method is used to check whether parent-child relation is allowed or not
    - This method ensure that invalid relation will not be stored in the database
    - **Input**: takes parent and child as input
    - **Return**: Integer value based on various conditions

    Following are the conditions:
    - Return 0 if the relation is already existing
    - Return 1 if user try to store child as parent if parent-child relation already exists
    - Return 2 if child already have 2 parents
    - Return 5 if the relation is allowed to store in the database

8. Public boolean recordChild(PersonIdentity parent, PersonIdentity child)
    - This method is used to add parent-child relation in the database.
    - The parent-child relation will be stored in “recordChild” table
    - A child can have only 2 parents
    - A parent can have multiple children.

- **Input**: PersonIdentity parent, PersonIdentity child
- **Return**: true if relation was stored in the database else return false

9. private int checkPartnering(int partner1, int partner2)
    - This method will check whether partnering will be possible between two individuals
    - **Input**: Partner id
    - **Input**: Partnet Id
    - **Return**: Integer value based on the various conditions

    Following are the conditions:

- Returns 0 if partner 1 is in 1st column and partner 2 is in 2nd column and status is partnering
- Returns 1 if partner 2 is in 1st column and partner 1 is in 2nd column and status is partnering
- Returns 2 if partner 1 is in 1st column and status is partnering
- Returns 3 if partner 2 is in 1st column and status is partnering
- Returns 4 if partner 1 is in 2nd column and status is partnering
- Returns 5 if partner 2 is in 2nd column and status is partnering
- Return 2 if partner 1 is in 1st column and partner 2 is in 2nd column and status is dissolution
    - Returns 3 if partner 1 is in 1st column and status is partnering
    - Returns 4 if partner 2 is in 1st column and status is partnering
    - Else return 2
- Return 8 if partner 2 is in 1st column and partner 1 is in 2nd column and status is dissolution
    - Returns 5 if partner 1 is in 2nd column and status is partnering
    - Returns 6 if partner 2 is in 2nd column and status is partnering
    - Else Return 2
- Else Return 11

10. public boolean recordPartnering(PersonIdentity partner1, PersonIdentity partner2)
    - This method records the symmetric partnering between two individuals
    - It will store the relation in the “recordPartneringDissolution” table
    - It will use checkPartnering(int partner1, int partner2) method to confirm whether partnering between two individual is possible or not
    - **Input**: PersonIdentity partner
    - **Input**: PersonIdentity partner
    - **Return**: true if the partnering between two individual is stored in the database else return false

11. public boolean recordDissolution(PersonIdentity partner1, PersonIdentity partner2)
    - This method records the symmetric dissolution between two individuals
    - It will store the relation in the “recordPartneringDissolution” table
    - It will use checkPartnering(int partner1, int partner2) method to confirm whether dissolution between two individual is possible or not
    - **Input**: PersonIdentity partner
    - **Input**: PersonIdentity partner
    - **Return**: true if the dissolution between two individual is stored in the database else return false

Methods that manage Media archive

1. private void addDatabaseMedia()
    - This private method is called when the object of the genealogy class is initialized. It basically creates the object of the data which is already present in the database table “mediaNameInfo”.
    - It will create the object that consists of media id, media file name and media file location and then it will store that object in the map as value. The key to that map would be the media id. This method is placed in the constructor of genealogy class so it will be called automatically**.**

2. public FileIdentifier addMediaFile(String fileLocation)
    - This method is used to add the media file in the database
    - It will add media in the table named “mediaNameInfo” in the database. It will take file location as input and from the file location this method extracts the file name. So, in the database the file name and location will be added, and the media id will be assigned automatically.
    - **Input**: File location
    - **Return**: **FileIdentifier** object

3. private boolean checkMediaAttribute(int id)
    - This method basically checks whether the media id exists or not.
    - It will check with the database table “mediaNameInfo” to arrive on a conclusion whether the media id exists or not.
    - **Input**: Media Id
    - **Return**: true if media id exists else false

4. public boolean recordMediaAttributes(FileIdentifier fileIdentifier, Map<String, String> attributes)
    - It will take Media information such as year, date and city as in the form of map as input. It will also take FileIdentifier object as parameter.
    - This method then store this information in “mediaAttributeInfo” table
    - **Input**: FileIdentifier Object, Map (Key as column name and value as information provided by user)
    - **Return**: true if data was stored in the database else return false.

5. public boolean peopleInMedia(FileIdentifier fileIdentifier, List<PersonIdentity> people)
    - This method basically connects the people with the given media
    - It will take single media file and a set of people that needs to be linked with that given media as input. It will then link each person to the given media file.
    - It will store this information in “mediaPersonRelation” table.
    - **Input**: FileIdentifier Object, List of persons
    - **Return**: true if people are connected to the media file in the system else return false

6. public boolean tagMedia(FileIdentifier fileIdentifier, String tag)
    - This method will store tag for the given media file
    - It will store tags in “mediaTag” table.
    - A media file can have multiple tags
    - **Input**: FileIdentifier, Tag
    - **Return**: true if tag was stored in the database else false

Reporting Function

1. PersonIdentity findPerson(String name)
    - It will find or locate the individual based on its Person name
    - **Input**: Person name
    - **Return**: PersonIdentity object
        - Returns null if person name is null or empty
        - Return null if the name of the person is not present list

2. public FileIdentifier findMediaFile(String name)
    - It will find or locate the media file based on its media file name
    - **Input**: Media File name
    - **Return**: FileIdentifier object
        - Returns null if person name is null or empty
        - Return null is the name of the person is not present list

3. public String findName(PersonIdentity id)
    - This method is used to get name of the individual using PersonIdentity object
    - Input: PersonIdentity object
    - Return: name of the individual
        - Return null if PersonIdentity object does not exists

4. public String findMediaFile(FileIdentifier fileId)
    - This method is used to get media file name using FileIdentifier object
    - Input: **FileIdentifier** object
    - Return: name of the media file
        - Return null if FileIdentifier object does not exists

5. public List<String> notesAndReferences(PersonIdentity person)
    - This method is used to get nodes and references of the individual in the same order in which they are added to the family tree
    - This method takes PersonIdentity object as input and returns it note and references. The nodes and references were stored in table the based-on timestamps.
    - This method make use of two tables “personNote” and “personReference”
    - **Input**: PersonIdentity object
    - **Return**: list of nodes and references of given PersonIdentity object in which they are added to the family tree

6. private boolean isTagExits(String tag)
    - This method will check whether entered tag exists or not
    - It will check with the database table “mediaTag” to arrive on a conclusion whether the tag exists or not.
    - **Input**: tag
    - **Output**: true if tag exists else false

7. public Set<FileIdentifier> findMediaByTag(String tag, String startDate, String endDate)
    - This method is used to find the set of media files that are linked to the given tag and fall within the give date range
    - Consider end date = null if end date is empty or null
    - Consider start date = null if start date is empty or null
    - **Input**: tag, startDate, endDate
    - **Output**: set of FileIdentifier objects that are linked to the give tags and falls within the give date range
        - Returns null if tag is null or empty
        - Returns null if the tag is not linked to any media

8. private boolean isLocationExists(String location)
    - This method will check whether entered location exists or not
    - It will check with the database table “personAttributeInfo” to arrive on a conclusion whether the location exists or not.
    - Input: tag
    - Output: true if location exists else false

9. public Set<FileIdentifier> findMediaByLocation(String location, String startDate, String endDate)
    - This method is used to find the set of media files that are linked to the given location and fall within the given date range
    - Consider end date = null if end date is empty or null
    - Consider start date = null if start date is empty or null
    - **Input**: location, startDate, endDate
    - **Output**: set of FileIdentifier objects that are linked to the give location and falls within the give date range
        - Returns null if location is null or empty
        - Returns null if the location is not linked to any media

10. public List<Integer> findIndividualsMedia(Set<PersonIdentity> people, String startDate, String endDate)
    - This method is used to find the set of media files that includes any of individual from the given list and falls within given range.
    - Consider start date = null if start date is empty or null
    - Consider end date = null if end date is empty or null
    - This method also removes the person which does not exists
    - **Input**: Takes the set of individuals as input, startDate and endDate
    - **Return**: list of media files which inclues individuals given in the list and falls within the given date range

11. public List<FileIdentifier> findBiologicalFamilyMedia(PersonIdentity person)
    - This method if used to find the set of media files that includes the specified person's immediate children.
    - This method uses the “recordChild” to find the person’s immediate children

- This method returns media files in ascending chronological order. It will add the list of media files with no dates ate the end of final list
- Input: PersonIdentity object
- Return: list of media files containing person's immediate children

12. public BiologicalRelation findRelation(PersonIdentity person1, PersonIdentity person2)
    - This method shows how two individuals are related to each other
    - This method returns the Biological Relation object that defines how two individuals are related
    - **Input**: PersonIdentity person1, PersonIdentity person2
    - **Return**: BiologicalRelation object

13. public Set<PersonIdentity> descendents(PersonIdentity person, Integer generations)
    - This method returns set of descendents of the person
    - This method make use of "recordChild" to find the person's descendents.
    - This method returns empty list if the person does not have descendents.
    - **Input**: PersonIdentity person, Integer generations
    - **Return**: Set of descendents

14. Public Set<PersonIdentity> ancestores(PersonIdentity person, Integer generations)
    - This method returns set of ancestores of the person
    - This method make use of "recordChild" to find the person's ancestores.
    - This method returns empty list if the person does not have ancestores.
    - **Input**: PersonIdentity person, Integer generations
    - **Return**: Set of ancestors

PersonIdentity class

This class is used to identify individuals from the family tree.

- It contains two variable Int personId and String personName. This class help to create the object of individual. This object helps use to identify the individual from the family tree

Methods:

1. **public int getId():** This method is used to get person id
2. **public String getName()**: This method is used to get person name

FileIdentifier Class

This class is used to identify file using fileLocation from media archive (Here we are taking file location instead of file name because name of two file can be same but location of two file is to be different)

- It contains three variable Int Id, String filename and String fileLocation. This class helps to create the object of each file. This object helps use to identify the file from the media archive.

Methods

1. **public int getId():** This method is used to get media id
2. **public String getFileName()**: This method is used to get file name
3. **public String getFileLocation()**: This method is used to get file location

BiologicalRelation Class

This class specifies the cousinship and the level of removal in a relation

- it Contains two variable int degreeOfRemoval and int degreeOfCousinship

Methods

1. **public int getDegreeOfRemoval():** This method is used to get drgree of removal
2. **public int getDegreeOfCousinship():** This method to get degreeOfCousinship

Data Structure

1. Map

- **Map<Integer, PersonIdentity> personMap:** This Map stores the Ids of the person in Keys and Object of the Person in Values. Object contains id and name of the person.

- **Map<Integer, FileIdentifier> fileMap:** This Map stares the ids of the file in keys and Objects of the File in Values. Objects contains id, name and location of the media

- **Map<String, String> personAttributes:** This Map stores the person attributes. The key is the database columns name such as “DOB”, “Gender” and “Occupation”. And the values will be the data entered by the user.

- **Map<String, String> mediaAttributes:** This Map stores the media file attributes. The key is the database columns name such as “Year”, “Date” and “Location”. And the values will be the data entered by the user.

2. Set

- **Set<PersonIdentity> people:** This set is used to store the multiple persons entered by the user. This set is then passed to findIndividualsMedia() method which is present in genealogy class.

- **Set<FileIdentifier> mediaFileName:** This set is used to store the media file name. This set is then returned by findMediaByTag(). This set contains those people who were there in the given media file.

- **Set<FileIdentifier> mediaFileName:** This set is used to store the media file location. This set is then returned by findMediaByLocation(). This set contains those people who were connected to the give location.

- **Set<PersonIdentity> ancestors:** This set is used to store all the ancestors of the person. This is then returned by ancestors() method that is present in genealogy class.

- **Set<PersonIdentity> descendents:** This set is used to store all the descendants of the person. This is then returned by descendents() method that is present in genealogy class.

3. List

- **List<PersonIdentity> people:** This list is used to store the person entered by the user. This list is then passed to peopleInMedia() method which is present in genealogy class.

- **List< FileIdentifier> mediaFileName:** This list is used to store the media file. This list is used in findIndividualsMedia() method. This list contains media files which includes set of people.

- **List<Integer> parent1**: This list is used to store the person1 ancestors. This list is used in findRelation() method

- **List<Integer> parent1**: This list is used to store the person2 ancestors. This list is used in findRelation() method

- **List<FileIdentifier> mediaFiles**: This list is used to store the media files of the in which a specified person immediate child appears. This list is used in findBiologicalFamilyMedia() method.

Database Tables

Following tables were created for family tree management

1. **personNameInfo:**

   This table stores the id and the name of the person. In this table data is added by addPerson() method. Here person id is primary key.

   Table Columns

personId int (Primary Key)	**personName** varchar(255)

2. **personAttributeInfo:**

   This table stores the person's attributes such as "date of birth", "date of death", "gender", "Occupation" and "person id". In this table data is added by recordAttributes() method. Here person id is foreign key from "personNameInfo" table.

   Table columns

personDOB varchar(255)	**personDOD** varchar(255)	**personGender** varchar(255)	**personOccupation** varchar(255)	**personId** int (foreign Key)

3. **personNote**

   This table stores the person's note. The person can have multiple notes. In this table data is added by recordNote() method. Here person id is foreign key from "personNameInfo" table. The lastupdated column records the time when the note of person is added in the table.

   Table columns

personNote varchar(255)	**personId** int (Foreign Key)	**lastUpdated**

4. **personReference**

This table stores the person's note. The person can have multiple references. In this table data is added by recordReference() method. Here person id is foreign key from "personNameInfo" table. The references can be location of birth certificate or web page that list when someone was graduated. The lastupdated column records the time when the note of person is added in the table

Table Columns

personReference varchar(255)	**personId** int (Foreign Key)	**lastUpdated**

5. **recordPartneringDissolution**

This table stores partnering and dissolution between two individuals. If there is partnering between two individuals then it will store it in the table and status is set as "partnering". And if there is dissolution between two individuals then it will update the relationship as "Dissolution". Here partner1 and partner2 is foreign key from "personNameInfo" table

Table Columns

partner_1 int (Foreign Key)	**partner_2** int (Foreign Key)	**relationship** varchar(255)

6. **recordChild**

This table stores the parent child relation. In this table the data is added by recordChild() method. Here parent is foreign key from "personNameInfo" table and child is foreign key from "personNameInfo" table.

Table Columns

parent int (Foreign Key)	**child** int (Foreign Key)

Following tables were created for media archive management

1. **mediaNameInfo**

This table stores the id name and location of the media. In this table data is added by addMediaFile () method. Here mediaid is the primary key.

Table Columns

mediaId int (Primary Key)	**mediaName** varchar(255)	**mediaLocation** varchar(255)

2. **mediaAttributeInfo**

   This table stores the media attributes such as “year”, “date”, “city” and “media id”. In this table data is added by recordMediaAttributes () method. Here media id is foreign key from “mediaNameInfo” table.

   Table columns

mediaYear varchar(255)	**mediaDate** varchar(255)	**mediaCity** varchar(255)	**mediaId** int (foreign Key)

3. **mediaPersonRelation**

   This table stores the people appear in the given media. In this table the data is added by peopleInMedia() method. Here media id is foreign key from “mediaNameInfo” table. And personId is foreign key from “personNameInfo” table.

   Table Columns

mediaId int (Foreign Key)	**personId** int (Foreign Key)

4. **mediaTag**

   This table stores the tag for the media. In this table the data is added by tagMedia() method. Here media id is foreign key from “mediaNameInfo” table.

   Table Columns

mediaTag varchar(255)	**mediaId** int (Foreign Key)

Assumptions

- For startDate and endDate, user must need to enter it in YYYY-MM-DD format.
- If parent child relation exists in the database then partnering between this is not possible
- If partnering between two individual exists in the database then parent-child between these two individuals is not possible.

Limitations

- The program doesn’t check for invalid months (month not in range of 1-12) or invalid date (date not in range of 1-31)

Files Submitted

1. Java files
    - Genealogy.java
    - FileIdentifier.java
    - BiologicalRelation.java
    - PersonIdentity.java
    - Main.java

2. login.properties file

3. Database files
    - Project_SQL (SQL file)
    - Database_ER_Diagram (PDF File)

4. Testing File
    - Test_Plan_and_Strategy (PDF file)

5. Final_Project_Doc (PDF)