***Library Management System:***

# Project description:

Our system is a file-based database system designed for employees **only** that accepts any number of employees, books or authors under one singleton class named Library (just one-branched library) which offers various of functionalities like: Adding/removing/updating displaying the 3 main members we mentioned before.

# The OOP Concepts Used:

1. Inheritance
2. Polymorphism
3. Aggregation (Relationship)
4. Overriding Method
5. Overloading Method
6. Interface

# The project design structure used:

1. User friendly GUI
2. Login form for users and owner/manager
3. Multi-threading
4. Network model
5. JDBC

# Running the program:

This program is created to carry out operations in a “Library Management System”. The employees can use this program to help them to manage the database of 5 different categories (Employees, Books, Authors, Members and Personnel).

Once the program is executed, a “Welcome to YMY Library” screen will be displayed in front of the user, when they login they will be asked to enter the “Admin User” and the “Password” thus, the “Library” screen will show the user 5 categories to choose from it “Employees, Books, Authors, Members and Personnel”. Every category has its own purpose that will be discussed in the next pages.

NOTE: in all the displayed screens the user has the ability to go back to the previous screens.

1. **The “Employees” category:**
   1. “Insert Employee” function: the user will be asked to enter the new employee information “Name, Age and ID”.
   2. “Delete Employee” function: the user will be asked to enter the employee ID that will be removed.
   3. “Display Employees” function: it will display all the existing employees in a table view that contains (ID, Name and Age)
   4. “Update Employee” function: the user will be asked to enter the employee ID to update an existing employee’s information.
2. **The “Books” category:**
3. “Insert Books” function: the user will be asked to enter the new book information “Book Title, Barcode, Author and Price”.
4. “Delete Books” function: the user will be asked to enter the book barcode that will be removed.
5. “Display Books” function: it will display all the existing books in a table view that contains (Book Title, Bar Code, Author and Price)
6. “Update Books” function: the user will be asked to enter the book barcode to update an existing book’s information.
7. **The “Authors” category:**
8. “Insert Author” function: the user will be asked to enter the new author information “Name and Age”.
9. “Delete Author” function: the user will be asked to enter the author’s name that will be removed.
10. “Display Author” function: it will display all the existing author in a table view that contains (Name and Age)
11. “Update Author” function: the user will be asked to enter the author’s name to update an existing author’s information.
12. **The “Members” category:**

“Display Member” function: it will display all the existing member in a table view that contains (Name, Age, Email, Phone number and ID).

\*this point will be discussed in the aggregation section too\*

1. **The “Personnel” category:**

“Display Personnel” function: it will display **all the existing members** of the library in a table view that contains all of their names.

\*this point will be discussed in the polymorphism section too\*

# Abstract Class:

Abstraction is a process of hiding the implementation details and showing only functionality to the user. A class which is declared as abstract is known as an abstract class. It can have abstract and non-abstract methods. It needs to be extended and its method implemented. It cannot be instantiated. To access the abstract class, it must be inherited from another class.

***Class Person*** (abstract class)***:***

Contains the main/common data fields of any character or person as: **fullName, age** knowing that it is an abstract class which you **can’t** create an instance (an object) out of it.

It is implemented as parent class for further usage/purposes with the child classes.

# The OOP Concepts Used:

1. **Inheritance:**

This is a special feature of Object-Oriented Programming in Java. It lets programmers create new classes that share some of the attributes of existing classes. This lets us build on previous work without reinventing the wheel. Inheritance is another labor-saving Java OOP concept. It works by letting a new class adopt the properties of another. We call the inheriting class a subclass or a child class. The original class is often called the parent. Inheritance represents the IS-A relationship which is also known as a parent-child relationship.

Terms used in Inheritance:

* Sub Class/Child Class: Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.
* Super Class/Parent Class: Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.

The “extends” keyword indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality. In the terminology of Java, a class which is inherited is called a parent or superclass, and the new class is called child or subclass.

***Class Librarian*** (child class: extends person)***:***

Contains all the data fields concerning the librarian (or employee) that will be created as an object of the class with the same data fields of parent’s class + other data fields as: **workID.**

***Class Author*** (child class: extends person)***:***

Contains all the data fields concerning the author that will be created as an object of the class with the same data fields of parent’s class + other data fields

as: **authorID.**

### Class Book:

Contains all the data fields concerning the book that will be created as an object with the data fields as: **bookTitle,barcode,authorName,price**

# Aggregation (Relationship):

Aggregation is a way to achieve Association. Aggregation represents the relationship where one object contains other objects as a part of its state. It represents the weak relationship between objects. It is also termed as a has

a relationship in Java. Like, inheritance represents the is-a relationship. It is another way to reuse objects. In Java, aggregation represents HAS-A relationship, which means when a class contains reference of another class known to have aggregation. The HAS-A relationship is based on usage, rather than inheritance. In other words, class A has-a relationship with class B, if class A has a reference to an instance of class B. The main advantage of using aggregation is to maintain code re-usability. If an entity has a relation with some other entity than it can reuse code just by referring that.

***Class Member*** (child class: extends person)***:***

In order to apply the “Aggregation” concept, the member class has the purpose of showing all the members(readers) that have some sort of subscription in the library. As meant to be, the library **has** some members

\*also shown the GUI sector in the documentation & will be explained in more details\*

# Polymorphism:

This Java OOP concept lets programmers use the same word to mean different things in different contexts. One form of polymorphism in Java is method overloading. That’s when different meanings are implied by the code itself. The other form is method overriding. That’s when the different meanings are implied by the values of the supplied variables. Polymorphism in Java works by using a reference to a parent class to affect an object in the child class.

Two more examples of polymorphism in Java are method overriding and method overloading:

In method overriding, the child class can use the OOP polymorphism concept to override a method of its parent class. That allows a programmer to use one method in different ways depending on whether it’s invoked by an object of the parent class or an object of the child class.

In method overloading**,** a single method may perform different functions depending on the context in which it’s called. That is, a single method name might work in different ways depending on what arguments are passed to it.

### Personnel Button:

In order to apply the “Polymorphism” concept, the personnel section has a purpose of showing all the meant members of the library (employees, authors, members) as they are all separate classes which inherits the same class (person) and with a press of button it initiate the same function (print) but with different outcomes (different categories names)

\*also shown the GUI sector in the documentation\*

# Overriding Method:

If subclass (child class) has the same method as declared in the parent class, it is known as method overriding in Java. In other words, if a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding. Method overriding is used to provide the specific implementation of a method which is already provided by its superclass. Method overriding is used for runtime polymorphism. The method must have the same name as in the parent class. The method must have the same parameter as in the parent class. There must be an IS-A relationship (inheritance).

# Overloading Method:

If a class has multiple methods having same name but different in parameters, it is known as Method Overloading. If we have to perform only one operation, having same name of the methods increases the readability of the program. Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as a (int, int) for two parameters, and b (int, int, int) for three parameters then it may be difficult for you as well as other programmers to understand the behavior of the method because its name differs. So, we perform method overloading to figure out the program quickly. Method overloading increases the readability of the program.

There are two ways to overload the method in java:

1. By changing number of arguments
2. By changing the data type

# Interface:

An interface is a reference type in Java. It is similar to class. It is a collection of abstract methods. A class implements an interface, thereby inheriting the abstract methods of the interface. Along with abstract methods, an interface may also contain constants, default methods, static methods, and nested types. Method bodies exist only for default methods and static methods. Writing an interface is similar to writing a class. But a class describes the attributes and behaviors of an object. And an interface contains behaviors that a class implements. Unless the class that implements the interface is abstract, all the methods of the interface need to be defined in the class.

An interface is similar to a class in the following ways:

* An interface can contain any number of methods.
* An interface is written in a file with a **.java** extension, with the name of the interface matching the name of the file.

However, an interface is different from a class in several ways:

* You cannot instantiate an interface.
* An interface does not contain any constructors.
* All of the methods in an interface are abstract.
* An interface cannot contain instance fields. The only fields that can appear in an interface must be declared both static and final.
* An interface is not extended by a class; it is implemented by a class.
* An interface can extend multiple interfaces.

The interface keyword is used to declare an interface. An interface is implicitly abstract. You do not need to use the abstract keyword while declaring an interface. Each method in an interface is also implicitly abstract, so the abstract keyword is not needed. Methods in an interface are implicitly public.

## Implementing Interfaces:

When a class implements an interface, you can think of the class as signing a contract, agreeing to perform the specific behaviors of the interface. If a class does not perform all the behaviors of the interface, the class must declare itself as abstract. A class uses the implements keyword to implement an interface. The implements keyword appears in the class declaration following the extends portion of the declaration (ex: author interface in our code)

# Multi-threading network model:

## ‘MainConnection’ Class:

Main controller of the network model we built, this class creates child of class Server and child from class Client and contains a Boolean variable named flag: boolean flag=true;

and by calling the method Login it initiates 2 threads, one for the server object (which will create the server firstly ) and the second thread made for the client.

To have a finalized network model working with more than one thread to send and receive data between client and server

## ‘Server’ Class:

Initializing few variables needed for the network model:

ServerSocket ss; Socket socket;

int port = 4333; ServerSocket ss; Socket socket;

DataInputStream dis; DataOutputStream dos; boolean flag=true;

Taking username and password throughout the ‘readUTF’ from the client class and save it into string to apply a certain restriction on the password with an if condition (password checked if it is less than or equal to 4 characters, If so it will be refused; else: will be accepted)

## ‘Client’ Class:

Initializing few variables needed for the network model:

Socket socket; int port = 4333;

DataInputStream dis; DataOutputStream dos;

Moving on to the Login method, this method will get username and password form the GUI and open connection with server for the purpose of sending these data to server to complete the login phase

# The project design structure:

**User friendly GUI using scene builder (java FXML):**

## A detailed description of the user interface scenes in a sequential order 1) WLECOME PAGE:

Introductory scene with the name of the system “Library Management System” with a Login button that switches to the next scene

## LOGIN PAGE:

Login scene where there are: username field and password field to let the user in the system, accompanied with 2 buttons:

-Login, to trigger the switching operation for the next scene

-Back, to go back to the previous scene

## MAIN PAGE:

Note: to avoid redundancy/repetition, each scene contains a back button that returns to the main page

Inside the main page we have several buttons that leads us to many other scenes that will be discussed in the next few lines as follows:

## (3.1) Employees Button:

The employees button leads us to the scene where the database info concerning all the employees that work in the library with all the details needed such as: name, ID, age

accompanied with 3 data fields where you can insert info for the purpose of adding or updating data of a certain employee

INSERT button: for adding new employee to the database

DELETE button: to delete certain employee’s info from the database using his ID only

UPDATE button: to update certain employee’s info \*also by keeping his ID with no changes as an identifier\*

DISPLAY button: displaying all employees’ info with the latest updates applied to the database

## (3.2) Books Button:

The books button leads us to the scene where the database info concerning all the books that exists in the library with all the details needed such as: title, barcode, author, price

accompanied with 4 data fields where you can insert info for the purpose of adding or updating data of a certain book

INSERT button: for adding new book to the database

DELETE button: to delete certain book’s info from the database using his barcode only

UPDATE button: to update certain book’s info \*also by keeping its barcode with no changes as an identifier\*

DISPLAY button: displaying all books’ info with the latest updates applied to the database

## (3.3) Authors Button:

The authors button leads us to the scene where the database info concerning all the authors that have books/contracts with the library with all the details needed such as: name, ID, age

accompanied with 3 data fields where you can insert info for the purpose of adding or updating data of a certain author

INSERT button: for adding new author to the database

DELETE button: to delete certain author’s info from the database using his ID only

UPDATE button: to update certain author’s info \*also by keeping his ID with no changes as an identifier\*

DISPLAY button: displaying all authors’ info with the latest updates applied to the

## (3.4) Members Button:

The members (readers that have subscription to the library) button leads us to the scene (table view) where the database info concerning all the members that have subscription/borrowed books from the library with all the details needed such as: name, ID, age, email, phone number

email for the purpose of applying the **Aggregation** relationship as explained that is a “Has-a” relationship.. in other words the library has members

## (3.5) Personnel Button:

The personnel (employees, authors, members) button leads us to the scene (table view) where the database info concerning all the pervious members we mentioned before that exist in our library by showing their name in divided columns for the purpose of applying the **Polymorphism** concept as explained before

## Exit PAGE:

By clicking the X-button or the exist button in the top bar most right the user receives a friendly confirmation message noticing him/her about the logout and adding a question as “do you want to save before exiting”

so the user is left with 2 options : “ok” or “Cancel”

## The project data storage mechanism:

**Database JDBC:**

We are using database version **"MYSQL Workbench 8.0"** for the sake of keeping permanent information of our system. JDBC stands for Java Database Connectivity, which is a standard Java API for database-independent connectivity between the Java programming language and a wide range of databases.

## The JDBC library includes APIs:

* Making a connection to a database.
* Creating SQL or MySQL statements.
* Executing SQL or MySQL queries in the database.
* Viewing & Modifying the resulting records.

The Concept of using it, fundamentally, JDBC is a specification that provides a complete set of interfaces that allows for portable access to an underlying database. And the interface we used: (Driver interface, Connection interface, Statement interface, ResultSet interface, SQLException interface)

## That can be found in java.sql package:

* **Connection:** represents the connection to the database.
* **Driver Manager:** obtains the connection to the database. (Another option is DataSource, used for connection pooling.)
* **SQLException:** handles SQL errors between the Java application and the database.
* **ResultSet and Statement:** model the data result sets and SQL statements.

## Java Database Connectivity with MySQL:

To connect java application with the mysql database, mysqlconnector.jar file is required to be loaded.

## Import the packages:

Requires that you include the packages containing the JDBC classes needed for database programming.

import java.sql.Connection; import java.sql.DriverManager; import java.sql.ResultSet; import java.sql.Statement; import java.sql.SQLException;

## Register the driver class:

The forName() method of Class class is used to register the driver class. This method is used to dynamically load the driver class. The driver class for the mysql database is com.mysql.jdbc.Driver.

Class.forName("com.mysql.cj.jdbc.Driver");

## Create the connection object:

The getConnection() method of DriverManager class is used to establish connection with the database.The connection URL for the mysql database is

("jdbc:mysql://localhost:3306/database\_name", "username", "password")

where jdbc is the API, mysql is the database, localhost is the server name on which mysql is running, we may also use IP address, 3306 is the port number, the database name, username: The default username for the mysql database is root and password: It is the password given by the user at the time of installing the mysql database.

conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/library", "root", "123456789");

## Create the Statement object:

The createStatement() method of Connection interface is used to create statement. The object of statement is responsible to execute queries with the database.

st = conn.createStatement();

## Execute the query:

The executeQuery() method of Statement interface is used to execute queries to the database. This method returns the object of ResultSet that can be used to get all the records of a table.

rs = st.executeQuery(query); while (rs.next()) {

member = new Member(rs.getString("fullName"), rs.getDouble("Age"), rs.getString("email"), rs.getString("phoneNum"), rs.getInt("memberID"));

MemberList.add(member);

System.out.println(rs.getInt("memberID") + ", " + rs.getString("fullName") + ", " + rs.getDouble("Age") + ", " +

rs.getString("email") + ", " + rs.getString("phoneNum")); }

## JDBC SQL Syntax:

Structured Query Language (SQL) is a standardized language that allows you to perform operations on a database, such as creating entries, reading content, updating content, and deleting entries.

We used the CRUD operation to imply it in our project (Create, Read, Update, and Delete) data from a database.

## Create Database:

The CREATE DATABASE statement is used for creating a new database.

CREATE DATABASE library;

## Create Table:

The CREATE TABLE statement is used for creating a new table.

create table library.member( memberID integer not null, fullName varchar(255),

age double not null, email varchar(255), phoneNum varchar(255), primary key (memberID)

);

## Insert Data:

The syntax for INSERT, represents the new data to appear in the respective columns.

insert into library.member values(1,'Adam Levine',35,'adamlevine@gamil.com','01205410038');

## Select Data:

The SELECT statement is used to retrieve data from a database. The Asterix "\*" means we select all the data in the table.

select \* from library.member;

# The Project UML Diagram:

