Workshop 2: Introduction to OOP Concepts

Julian Santiago Florez Castañeda

Juan Jose Caicedo Medellin

Object Oriented Programming

- **Library Management System**:

Main Goals: get what books are available or not, know the data from this books, if a book is lent, know who have it and when does they need to return the book, also the customers would like to know what librarian have a section of the library in charge

Book: Title, Author, Pages, ID, ISBN, Editorial, Collection, Category, Location, Available, Section

Get if book is available for lent; know basic data from the book such like titles, authors, pages; Search the book by category, by location, by section

Member: Name, ID, Borrowed Books, Email

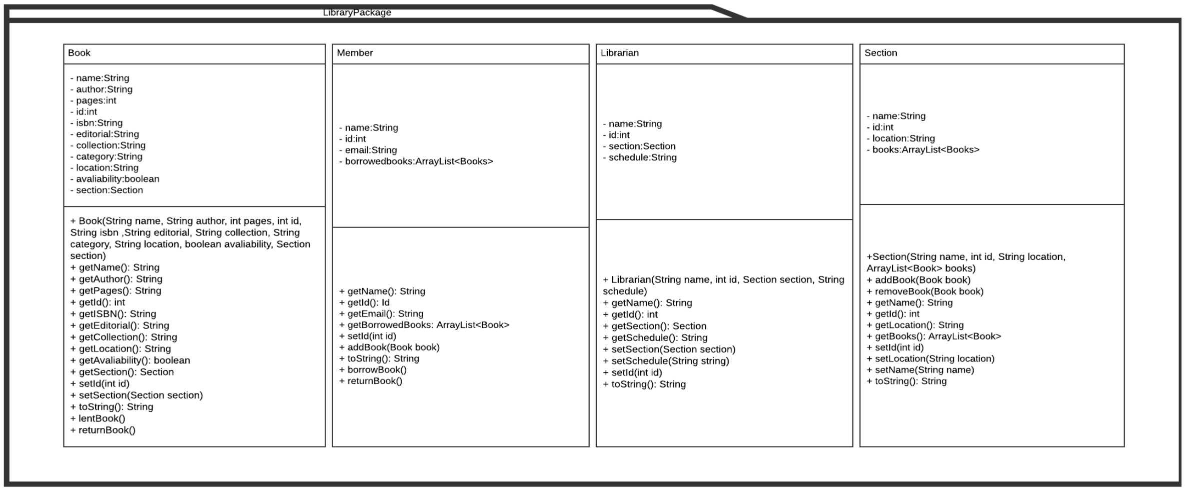
Get what books does the member have, when does the member have to return the book

Librarian: Name, ID, Section, Schedule

Get what section a librarian have in charge, get the work schedule of a librarian

Section: Categories,ID, Location, Books

Get what books are in this section, where is the section located, what categories identify this section



- **Online Shopping Platform:**

Main Goals:provide a seamless shopping experience for customers by managing their shopping carts, processing orders, and handling payments securely.

Customer: Name, ID, email, address

Get customer details like name and ID, let the user add orders

Product: Name, ID, Price, Category, Stock

Retrieve product information including name, code, price, category, and available stock. Update the product stock of price

Cart: Customer, Products

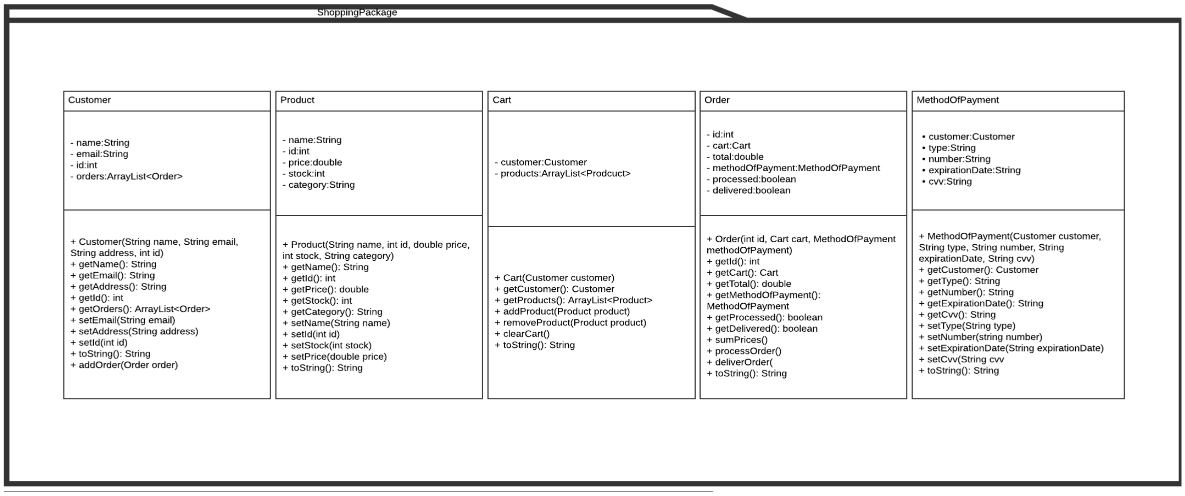
Track the products added to a customer's shopping cart.

Order: ID, Cart, Total Price, Delivery Status,Process Status

Store order details such as ID, Cart and the customer the order belongs to, process and deliver the orders

Payment Method: Customer, Type, number, expirationDate, cvv

Manage customer payment methods, including method type and associated card or bank details. Update if necessary



- **School Management System**:

Managing educational information efficiently, including storing student and teacher data, tracking schedules, classrooms, grades, and subjects taught.

Student: Name, Grades, Classroom, ID, ScheduleList

Store student name and ID, the classroom they are, the schedule and classroom where the student have their classes, the grades of a specific subject

Teacher: Name, Subject, ClassroomList, ID, ScheduleList

Store teacher name and ID, the classroom where they have to teach, the subjects that they teach, and the schedule

Classroom: StudentList, ID

Store all the students in the classroom, and the class number

Grade: Student, Subject, Note

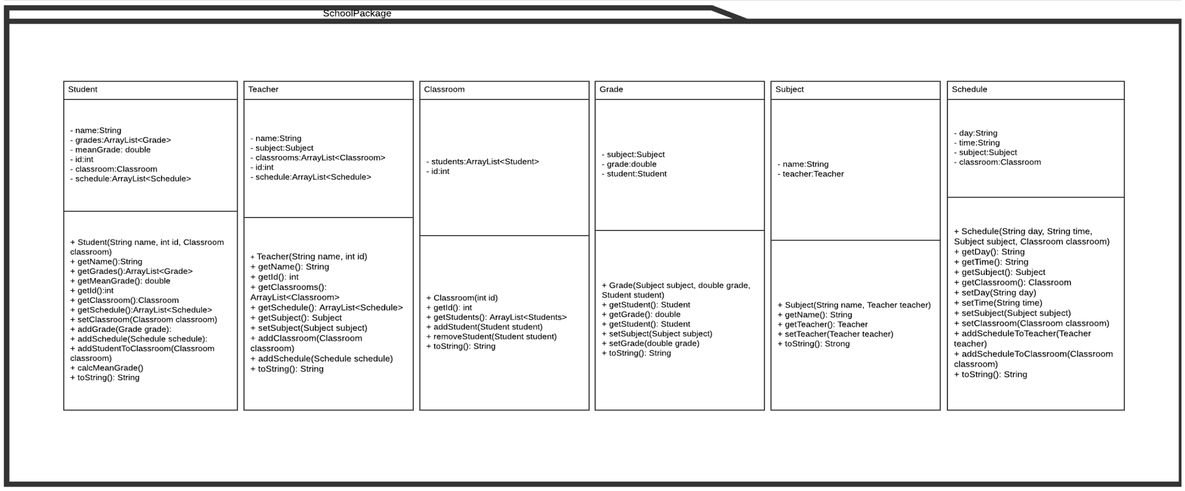
Get the grade that a student have for a specific subject

Subject: Teacher, Name

Store a subject Name and what teacher teach that subject

Schedule: Day, Hour, Subject, Classroom

Get the day, hour, location and subject for a class


SchoolPackage
Student
- name:String
- grades:ArrayList<Grade>
- meanGrade: double
- id:int
- classroom:Classroom
- schedule:ArrayList<Schedule>
+ Student(String name, int id, Classroom classroom)
+ getName():String
+ getGrades():ArrayList<Grade>
+ getMeanGrade(): double
+ getId():int
+ getClassroom():Classroom
+ getSchedule():ArrayList<Schedule>
+ setClassroom(Classroom classroom)
+ addGrade(Grade grade):
+ addSchedule(Schedule schedule):
+ addStudentToClassroom(Classroom classroom)
+ calcMeanGrade()
+ toString(): String
Teacher
- name:String
- subject:Subject
- classrooms:ArrayList<Classroom>
- id:int
- schedule:ArrayList<Schedule>
+ Teacher(String name, int id)
+ getName(): String
+ getId(): int
+ getClassrooms(): ArrayList<Classroom>
+ getSchedule(): ArrayList<Schedule>
+ getSubject(): Subject
+ setSubject(Subject subject)
+ addClassroom(Classroom classroom)
+ addSchedule(Schedule schedule)
+ toString(): String
Classroom
- students:ArrayList<Student>
- id:int
+ Classroom(int id)
+ getId(): int
+ getStudents(): ArrayList<Students>
+ addStudent(Student student)
+ removeStudent(Student student)
+ toString(): String
Grade
- subject:Subject
- grade:double
- student:Student
+ Grade(Subject subject, double grade, Student student)
+ getStudent(): Student
+ getGrade(): double
+ getStudent(): Student
+ setSubject(Subject subject)
+ setGrade(double grade)
+ toString(): String
Subject
- name:String
- teacher:Teacher
+ Subject(String name, Teacher teacher)
+ getName(): String
+ getTeacher(): Teacher
+ setTeacher(Teacher teacher)
+ toString(): Strong
Schedule
- day:String
- time:String
- subject:Subject
- classroom:Classroom
+ Schedule(String day, String time, Subject subject, Classroom classroom)
+ getDay(): String
+ getTime(): String
+ getSubject(): Subject
+ getClassroom(): Classroom
+ setDay(String day)
+ setTime(String time)
+ setSubject(Subject subject)
+ setClassroom(Classroom classroom)
+ addScheduleToTeacher(Teacher teacher)
+ addScheduleToClassroom(Classroom classroom)
+ toString(): String

- **Restaurant Management System**:

Manage a restaurant's operations by organizing menu items with details like name, description, price, category, and availability. It also records orders with their respective details such as order ID, table number, items ordered, total price, and status, while also managing table information like number, capacity, status, and customers. Additionally, it maintains records of customer details including name, ID, and contact information, along with staff members' information like name, ID, role, and schedule

Menu: Item Name, Description, Price, Category, Availability

Keep track of menu items with details like name, description, price, category, and availability.

Order: Order ID, Table, Items, Total Price, Status

Record order information, including order ID, customer ID, table number, ordered items, total price, and order status.

Table: Table Number, Capacity, Status, Customers

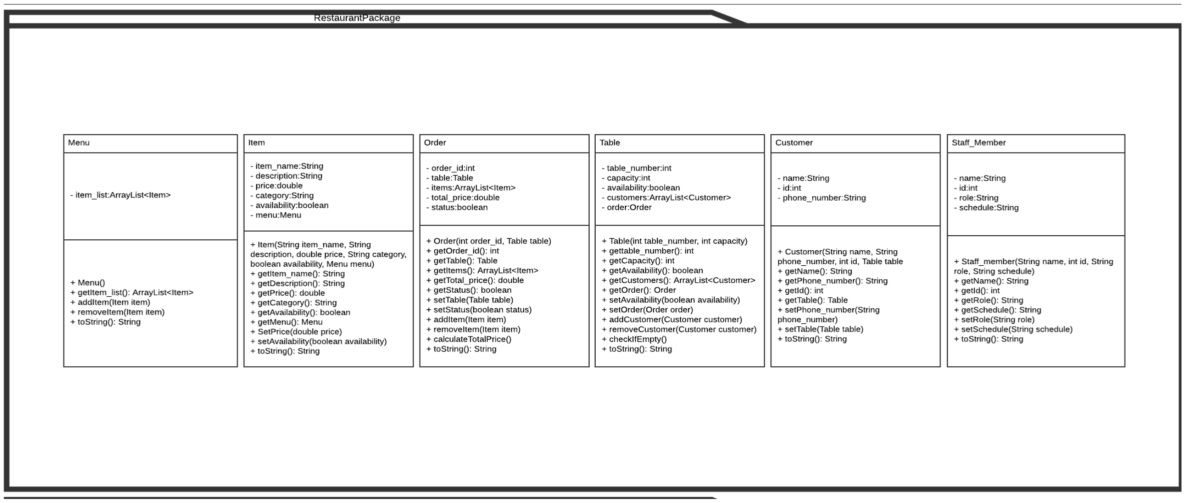
Manage tables by storing information such as table number, capacity, and current status.

Customer: Name, ID, Contact Number

Maintain customer details, including name, ID, and contact information.

Staff Member: Name, ID, Role, Schedule

Keep information about staff members, including name, ID, role, and work schedule.



- **Fitness Tracker Application**:

Manage user fitness data, storing user details such as name, ID, and personal information, along with tracking individual exercises including name, type, duration, and calories burned. It records workout sessions, comprising workout ID, user, list of exercises, total duration, and calories burned, while also facilitating the setting and tracking of fitness goals, specifying the user, target type, target value, and deadline.

User: Name, ID, Contact Number, Weight, Height, Age, Body Mass Index (BMI)

Store user details, including username, ID, and personal information.

Exercise: Exercise Name, Type, Duration, Calories Burned

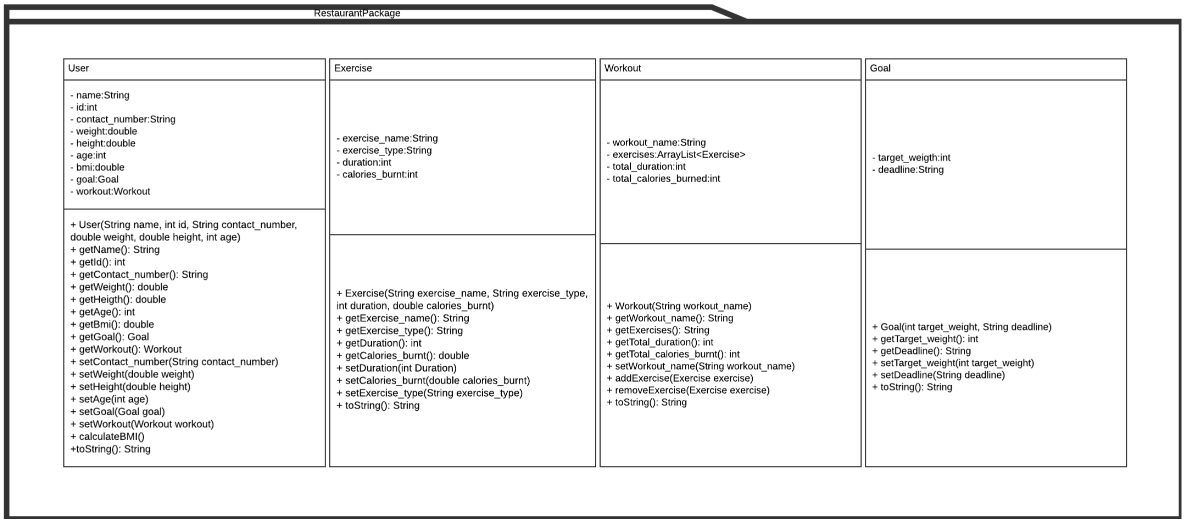
Keep track of individual exercises with details such as name, type, duration, and calories burned.

Workout: Workout Name, User, Exercises, Total duration, Total calories

Record workout sessions, including workout name, user, list of exercises, total duration, and calories burned.

Goal: User, Target Weight, Deadline

Set and track fitness goals, specifying the user, target weight and deadline.



```
User
- name:String
- id:int
- contact_number:String
- weight:double
- height:double
- age:int
- bmi:double
- goal:Goal
- workout:Workout
+ User(String name, int id, String contact_number,
double weight, double height, int age)
+ getName(): String
+ getId(): int
+ getContact_number(): String
+ getWeight(): double
+ getHeigth(): double
+ getAge(): int
+ getBmi(): double
+ getGoal(): Goal
+ getWorkout(): Workout
+ setContact_number(String contact_number)
+ setWeight(double weight)
+ setHeight(double height)
+ setAge(int age)
+ setGoal(Goal goal)
+ setWorkout(Workout workout)
+ calculateBMI()
+toString(): String
```

```
Exercise
- exercise_name:String
- exercise_type:String
- duration:int
- calories_burnt:int
+ Exercise(String exercise_name, String exercise_type,
int duration, double calories_burnt)
+ getExercise_name(): String
+ getExercise_type(): String
+ getDuration(): int
+ getCalories_burnt(): double
+ setDuration(int Duration)
+ setCalories_burnt(double calories_burnt)
+ setExercise_type(String exercise_type)
+ toString(): String
```

```
Workout
- workout_name:String
- exercises:ArrayList<Exercise>
- total_duration:int
- total_calories_burned:int
+ Workout(String workout_name)
+ getWorkout_name(): String
+ getExercises(): String
+ getTotal_duration(): int
+ getTotal_calories_burnt(): int
+ setWorkout_name(String workout_name)
+ addExercise(Exercise exercise)
+ removeExercise(Exercise exercise)
+ toString(): String
```

```
Goal
- target_weigth:int
- deadline:String
+ Goal(int target_weight, String deadline)
+ getTarget_weight(): int
+ getDeadline(): String
+ setTarget_weight(int target_weight)
+ setDeadline(String deadline)
+ toString(): String
```

https://lucid.app/lucidchart/628bd372-6015-4e1c-8c8a-3d9fe56ad42b/edit?view_items=TXGcvA8~SPiO&invitationId=inv_4b585bec-3411-44e8-a46a-3569514c7121

5.
Encapsulation:

Encapsulation in the context of object-oriented programming is a structural principle akin to packaging the attributes and associated methods of a class into a cohesive unit. It operates on the premise of restricting direct access to internal details while exposing only essential functionalities through well-defined interfaces.

Related Terms:

1. **Data Hiding:** This involves the restriction of access to certain attributes by designating them as private. The intent is to prevent unauthorized external entities from directly modifying or accessing sensitive internal data.

2. **Abstraction:** Encapsulation facilitates abstraction by presenting a simplified interface to users, shielding them from the intricacies of the underlying implementation. Users interact with the class through a high-level interface without needing to comprehend the internal complexities.

3. **Modularity:** Encapsulation promotes modularity by encapsulating the implementation details within a class. Each class serves as a modular unit, allowing for independent development, maintenance, and modification without affecting the broader system.

4. **Flexibility:** The concept of encapsulation enables flexibility by confining modifications within the class. Internal changes can occur without impacting external code as long as the public interface remains consistent, providing a robust foundation for adapting to evolving requirements.

5. **Maintainability:** The encapsulation principle significantly contributes to maintainability. The organization of attributes and methods within a class simplifies debugging, updates, and modifications. Changes can be localized to the relevant class, enhancing the overall maintainability of the codebase.

Importance of Defining Attributes as Private or Public:

Determining whether attributes should be private or public holds considerable significance, particularly in an academic context. Privatizing attributes serves as a means of enforcing encapsulation principles, restricting direct access to internal components. This intentional restriction ensures the integrity of the class, preventing unintended modifications that could compromise the overall functionality.

In an academic setting, adherence to data-hiding principles through the careful designation of private or public attributes contributes to the development of robust, well-structured code. It emphasizes the importance of understanding and implementing encapsulation to instill best practices in software design, fostering clarity, reliability, and maintainability in programming endeavors.