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

Book	Member	Librarian	Section
- name-String - author/String - pages:int - id:int - id:i	- name:String - idint - email:String - borrowedbooks:ArrayList <books></books>	- name String - i-id-int - section: Section - schedule: String	- name String - id.int - location.String - books ArrayList <books></books>
+ Book(String name, String author, int pages, int id, String isbn , String editorial, String collection, String category), String jocation, boolean availability, Section editory, String + getName(), String + getAuthor(), String + getAuthor(), String + getBash(); String + getBash(); String + getEditorial(); String + getEditorial(); String + getCation(); String + getCation(); String + getCation(); String + getAvailability(); boolean + getAvailability(); boolean + getAvailability(); boolean + getGetCation() + setSection(Section section) + setSection(Section section) + lentBook() + lentBook()	+ getName(): String + get(d): Id + getEmail(): String + getEmail(): String + getBornowetBooks: ArrayList <book> + set(d): Id) + set(d): Id) - suffbook() Sook book) + bornowBook() + returnBook()</book>	+ Librarian(String name, int id, Section section, String schedule) + getName() String + get(d): int + getSection(): Section + getSchedule(): String + setSection()Section section) + setSchedule(String string) + setSichedule(String string) + setSichedule(String string) + setSichedule(String string)	*Section(String name, Int id, String location, ArrayList*Book> books) * addBook[dook book] * removeBook[Book book] * removeBook[Book book] * getName(): String * getBd(): int * getLocation(): String * getBooks(): ArrayList*Book> * setId(int id): Scring location) * setId(int id): Scring location) * setName(String location) * toString(): String * toString(): String

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

Customer	Product	Cart	Order	MethodOfPayment
- name:String - email:String - idint - orders:ArrayList <order></order>	- name:String - id-int - price double - stockint - category:String	- customer:Customer - products:ArrayList <prodcuct></prodcuct>	- id:int - cart:Cart - total:double - methodOfPayment:MethodOfPayment - processed:boolean - delivered:boolean	customer:Customer type:String number:String expirationDate:String cvv:String
+ Customer(String name, String email, String address, int id) + getName(): String + getEmail(): String + getEddress(): String + getAddress(): String + getOdress(): GetOdress(): String + setAddress(String address) + setAddress(String address) + setOdress(): String + addCrder(Order order)	+ Product(String name, int id, double price, int stock, String category) + genName(): String + getPrice(): double + gestPrice(): double + gestStock(): int + getCategory(): String + setName(String name) + setId(Int id) + setStock((nt stock) + setPrice(double price) + soString(): String	+ Cart(Customer customer) + getCustomer(): Customer + getProducts): ArrayList-Product> + addProduct(Product product) + removeProduct(Product product) + clacificant() + clacificant() + toString(): String	+ Order(int id, Cart cart, MethodOfPayment methodOfPayment) + getid(f): int + geticart): Cart + getional(f): double + getional(f): double - getional(f): d	HethodOlPaymert(Customer customer, String type. String rumber, String rumber, String expirationDate, String cov) + getType(). String + getExpirationDate(). String + getExpirationDate(). String + getExpirationDate(). String + getExpirationDate(). String + setType(String type) + setNumber(string rumber) + setExpirationDate(String expirationDate) + setCvV(String cw + steString String

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	Teacher	Classroom	Grade	Subject	Schedule
- name:String - grades:ArrayList <grade> - meanGrade: double - id:int - classroom:Classroom - schedule:ArrayList<schedule></schedule></grade>	- name:String - subject:Subject - classrooms:ArrayList <classroom> - id:int - schedule:ArrayList<schedule></schedule></classroom>	- students:ArrayList <student> - id:int</student>	- subject-Subject - grade double - student-Student	- name:String - teacher:Teacher	- day/String - tme/String - expect/Subject - classroom/Classroom
+ Student(String name, int id, Classroom classroom) (String) - getCrade() ArrayList-Crade> - setClassroom(Classroom classroom) - addStchade() Grade(s chade) - addStchade() ArrayList-Crade) - addStchade() - addStcha	- Teacher(String name, int id) - getName(). String - getN): getN	+ Classroom(nt is) + get(d), int + get(d), int + get(d), int + get(d), int + get(d), Arrayl, is+Studento- + get(d), int + removeStudent(Student student) + toString(): Streig **TemoveStudent(Student student) + toString(): Streig	* Cirade(Subject subject, double grade, Student studend) * qetStudend(). Student * qetStudend(). Student * qetStudend(). Student * extStubjectStudend(). Student * extStubjectStudend(). Student * extStubjectStudend(). Student * extStubjectStudend(). Student * extStubjectStudent subject) * extStubjectStudent subject) * extStudend(). Student * extStudend(). Student * extStudend(). Student	Subject(Subjeyname, Teacher teacher) pethamet): Strip pethamet pethamet	Schedule(String day, String time, Subject subject, Classroom) (eDMy): String (eDMy): String (eDMy): String (eDM):
			JL		

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.

Menu	Item	Order	Table	Customer	Staff_Member
- item_list:ArrayList <item></item>	item_name:String - description:String - price:double - category:String - availability:boolean - menu:Menu	- order_id:int - table:Table - items:ArayList tem - total_price:double - status:boolean	- table_number.int - capacity.int - availability.boolean - customers:ArrayList <customer> - order:Order</customer>	- name:String - id:int - phone_number:String	- name:String - iddint - role:String - schedule:String
+ Menu() + gettlem_list(): ArrayList <tlem> + additem(tlem item) + removetlem(tlem item) + toString(): String</tlem>	Hemi(String item_name, String description, double price, String category, boolean availability, Menu menu) egettem_name() String egettem_name() String egetCategory() String egetCategory() String egetCategory() String egetAvailability(; boolean egetMenu() Menu essetPrec(double price) esetAvailability(boolean availability) essetAvailability(poolean availability)	+ Order(mt order_id, Table table) + getCrober_id(): int - getTable(): Table - getTable(): South - getStatus(): South - getStatus(): South - getStatus(): South - setStatus(): South - setStatus(): South - additem()(em item) - calculate (order-id) - calculate (order-	+ Table(int table_number, int capacity) + getable_number(): int - getCapacity(): int booksn - getCapacity(): int b	+ Customer(String name, String phone, number, int id, Table table + gerklame(): String + gerklame(): String + gerd(): int gerklame(): String + gerd(d): int gerklame(): Table + seetPhone, number(String phone, number) + setThabe(rathet table) + to String(): String	Staff_member(String name, int id, String role, String schedule) Tools String schedule) settle String spettide, String spettide, String spettide(String) settle(String) settle(String) settle(String) settle(String) settle(String) settle(String)

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	Exercise	Workout	Goal
- name-String - idinit - contact number:String - weight-double - height-double - age int - burit-double - goal Goal - goal Goal - workout. Workout + User(String name, int id, String contact number,	- exercise_name: String - exercise_type: String - duration;nt - calories_burnt:int	- workout_name:String - exercises:ArrayList=Exercise> - total_duration:int - total_calories_burned.int	- target_weigth-int - deadline-String
+ User(sving name, in d., sving contact_number, double weight, double height, int age) + genName(): String geld(): fit geld():	+ Exercise(String exercise_name, String exercise_type, int duration, double calories_burn() + yetExercise_name(). String + yetExercise_type(). String + yetDuration(). Professional to yetDuration(). Professional to yetDuration(). Professional to yetDuration(). Duration(). He setDuration() thoration() + setDuration() thoration(). He setDuration() thoration() + setDuration(). Professional to yetDuration(). Professional to yetDuration(). Professional to yet	Workout(String workout_name) + getVorkout_name). String + getExercises(). String - getTotal_duration(): int - getTotal_duration(): int - setVorkout_name(): first(): int - setVorkout_name(): first(): int - setVorkout_name(): first(): workout_name() - removeExercise(Exercise exercise) + removeExercise(Exercise exercise) + toString(): 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.