Jun 24, 2024 | JEDI - Batch 1 | 21st June to 2nd July

Notes

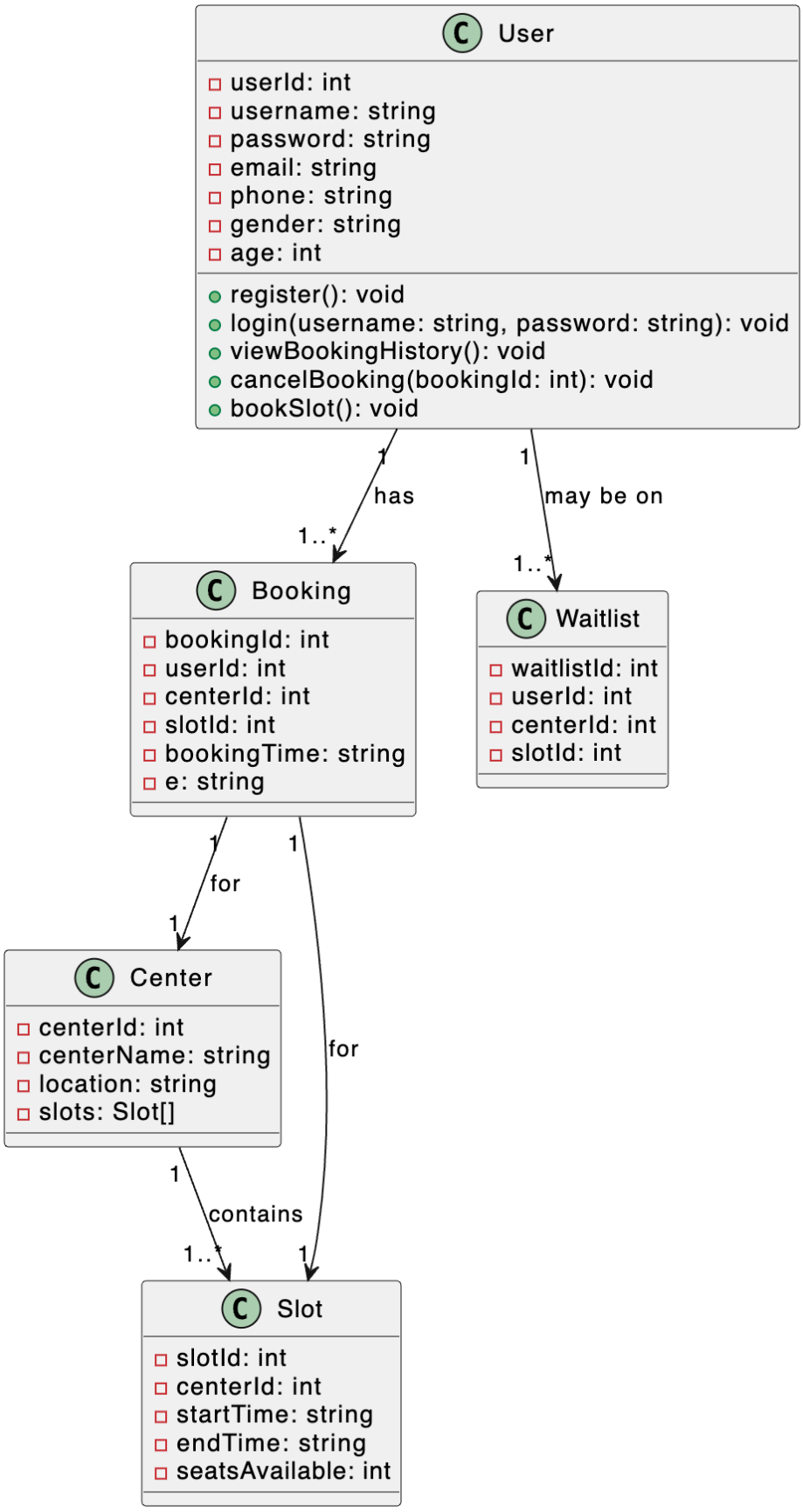
-

Action items

- [ ] Approval Docs for the gym owners
- [ ] Registration by customers
- [ ] Design UML Artefacts
- [ ] Make Lucid Chart
    - [ ] Use case
    - [ ] Activity
    - [ ] Class Diagram

Relationships between use cases:

1. Include relations
2. Extend
3. inherit


User
userId: int
username: string
password: string
email: string
phone: string
gender: string
age: int
register(): void
login(username: string, password: string): void
viewBookingHistory(): void
cancelBooking(bookingId: int): void
bookSlot(): void
1
has
1..*
1
may be on
1..*
Booking
bookingId: int
userId: int
centerId: int
slotId: int
bookingTime: string
e: string
Waitlist
waitlistId: int
userId: int
centerId: int
slotId: int
1
for
1
1
for
Center
centerId: int
centerName: string
location: string
slots: Slot[]
1
contains
1..*
1
Slot
slotId: int
centerId: int
startTime: string
endTime: string
seatsAvailable: int

Database Schema:

- Gym Center Schema -> stores the data about available gyms
- Slot Schema: For a gym id stores available morning and evenings slots
- User Schema: Registered user table-> unique id
- Booking : for a user id, stores the booking done-> a user can only book one slot

UML diagram classes:

- User
- Customer
- GymOwner
- Admin
- Gym
- PlatformAdmin
- Booking
- Wait list
- Slot

Types of users: Customer, Gym Owner, Admin: different privileges

Explanation of the UML Diagram:

1. **User Class**:
   - Represents a user of the FlipFit application.
   - Attributes include `userId`, `username`, `password`, `email`, `phone`, `gender`, `age(?)`.
   - Methods include `register()`, `login(username, password)`, `viewBookingHistory()`, and `cancelBooking(bookingId)`, `bookSlot()`
2. **Booking Class**:
   - Represents a booking made by a user for a particular slot at a center.
   - Attributes include `bookingId`, `userId`, `centerId`, `slotId`, `bookingTime`
3. **Center/Gym Class**:
   - Represents a gym center.
   - Each center has multiple slots (instances of `Slot`).
   - Contains basic information about the center (`centerId`, `centerName`, `location`, `slots`).
4. **Slot Class**:
   - Represents a specific time slot at a gym center.
   - Attributes include `slotId`, `centerId`, `startTime`, `endTime`, `seatsAvailable`.
5. **Waitlist Class**:

    - Represents a waitlist entry when a slot is fully booked.
    - Attributes include `waitlistId`, `userId`, `centerId`, `slotId`.

6. **Payment Gateway**
   Standard payment gateway
   Take transaction details and show the final cost

Relationships:

- **User** has a **Booking** (one-to-many): A user can have multiple bookings.
  User has three subclasses : Customer, GymOwner, Flipkart admin
- **Center** has many **Slot** instances (one-to-many): Each center can have multiple time slots.
- **Booking** relates to **Center** and **Slot** (many-to-one): Each booking is associated with a specific center and slot.
- **Users** may be on **Waitlist** (one-to-many): When a slot is fully booked, multiple users can be on the waitlist for that slot.

Classes and their attributes/methods:

User Class:
- **Attributes:**
  - `user_id`: Unique identifier for each user.
  - `username`: Name of the user.
  -password : password of the user
  -confirm password : password of the user
  - `email`: Email address of the user.
  - `phone_number`: Contact number of the user.

- **Methods:**
  - `register()`: Registers a new user with the gym platform.
  - `update_profile()`: Allows the user to update their profile information.
  - `view_classes()`: Retrieves and displays the classes available to the user.
  - `book_class(class_id)`: Books a class identified by `class_id`.
  - `cancel_booking(booking_id)`: Cancels a previously booked class.

Customer Class (Inherits from User):
- **Attributes:**

  - Inherits all attributes from `User`.
  - `payment_info`: Payment details of the customer (stored securely).

- **Methods:**
  - Inherits methods from `User`.
  - `make_payment(amount)`: Processes a payment for services or products.
  - `view_membership_details()`: Displays details of the current membership.

GymOwner Class:
- **Attributes:**
  - `owner_id`: Unique identifier for the gym owner.
  - `gym_name`: Name of the gym owned.
  - `location`: Physical location of the gym.
  - `employees`: List of employees (trainers, staff) associated with the gym.

- **Methods:**
  - `add_trainer(trainer_info)`: Adds a new trainer to the gym.
  - `remove_trainer(trainer_id)`: Removes a trainer from the gym.
  - `manage_schedule()`: Manages the gym's class schedule.
  - `view_revenue()`: Displays revenue details and analytics.

Admin Class:
- **Attributes:**
  - `admin_id`: Unique identifier for the admin.
  - `role`: Role description (e.g., platform admin, gym admin).

- **Methods:**
  - `approve_trainer(trainer_id)`: Approves a trainer to start working.
  - `manage_users()`: Manages user accounts (e.g., suspend, activate).
  - `generate_reports()`: Generates various reports (e.g., membership statistics, revenue).

Gym Class:
- **Attributes:**
  - `gym_id`: Unique identifier for each gym.
  - `name`: Name of the gym.
  - `location`: Location of the gym.
  - `classes_available`: List of classes available at the gym.

- **Methods:**
  - `add_class(class_info)`: Adds a new class to the gym's schedule.
  - `remove_class(class_id)`: Removes a class from the gym's schedule.
  - `view_schedule()`: Displays the current class schedule.

PlatformAdmin Class:

- **Attributes:**
  - `admin_id`: Unique identifier for the platform admin.

- **Methods:**
  - `manage_gyms()`: Manages gyms registered on the platform (add, remove).
  - `manage_admins()`: Manages admin accounts (add, remove, update roles).

Booking Class:
- **Attributes:**
  - `booking_id`: Unique identifier for each booking.
  - `user_id`: ID of the user who made the booking.
  - `class_id`: ID of the class booked.
  - `booking_time`: Date and time when the booking was made.

- **Methods:**
  - `confirm_booking()`: Confirms the booking and updates relevant records.
  - `cancel_booking()`: Cancels a booking and updates availability.

WaitList Class:
- **Attributes:**
  - `class_id`: ID of the class for which the waitlist is maintained.
  - `waitlist_entries`: List of users on the waitlist for the class.

- **Methods:**
  - `add_to_waitlist(user_id)`: Adds a user to the waitlist for a class.
  - `remove_from_waitlist(user_id)`: Removes a user from the waitlist.

Slot Class:
- **Attributes:**
  - `slot_id`: Unique identifier for each time slot.
  - `class_id`: ID of the class associated with the slot.
  - `start_time`: Start time of the slot.
  - `end_time`: End time of the slot.

- **Methods:**
  - `get_available_slots(date)`: Retrieves available slots for a given date.
  - `reserve_slot(slot_id)`: Reserves a slot for a user.

Notes:
- These classes provide a basic outline and can be expanded based on specific requirements and functionalities of the gym platform.
—----------

- Inheritance is utilized where applicable (e.g., `Customer` inherits from `User`) to capture relationships and common attributes/methods.
- Additional attributes and methods may be necessary depending on the complexity and features of the gym platform application.

Assumptions:
slotId is Unique

Diagram Links:

Class diagram Lucidchart
Activity diagram Lucidchart
Use case diagram Lucidchart

Jun 25, 2024 | JEDI - Batch 1 | 21st June to 2nd July

Git & GitHub

1. There are various project management tools which are using to get manage the project lifecycle as well as versioning.
   1.1 BitBucket -> partially paid
   1.2 CVS (Concurrent version system) -> Paid
   1.3 Tortoise SVN -> paid
   1.4 GitHub -> free

2. The equation of git is
   Git = core git (VCS) (git scm client) + github (remote repo/cloud repo) (need github cloud account)

3. Using git bash there are various git life cycle command which would be used to manage entire the enterprise project

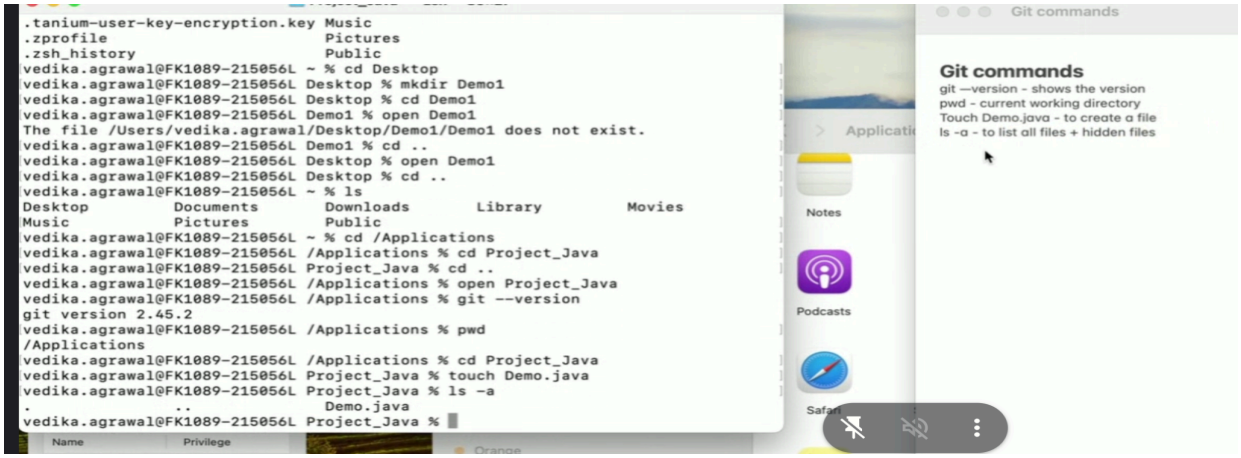
- Git add, git pull, git push,git commit, git status, git init, git log, git show, git remove, etc.

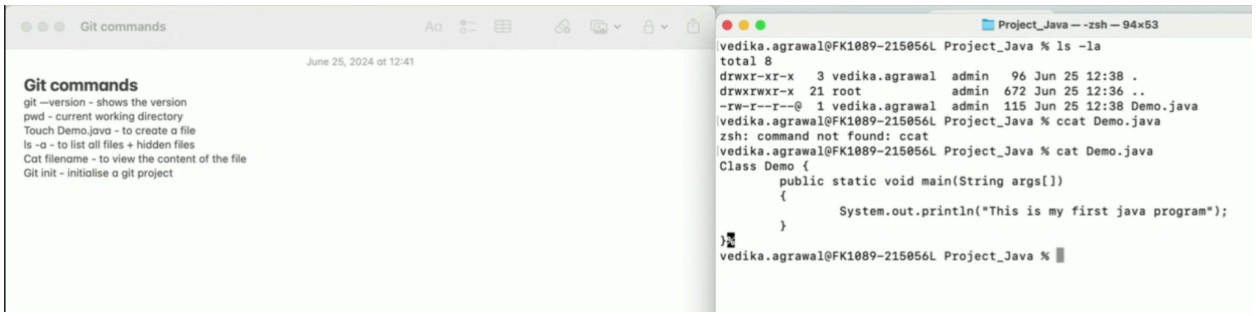
Git Remote**

4. In git projects management, git env is separated in 3 parts
   - Working directory-> staging area -> repository (with .git folder)

Demo:

1. Demo of git using core java project using command line
2. Demo of git using java project using sts or intelliJ






```
vedika.agrawal@FK1089-215056L Project_Java % ls -la
total 8
drwxr-xr-x   3 vedika.agrawal  admin   96 Jun 25 12:38 .
drwxrwxr-x  21 root            admin  672 Jun 25 12:36 ..
-rw-r--r--@  1 vedika.agrawal  admin  115 Jun 25 12:38 Demo.java
vedika.agrawal@FK1089-215056L Project_Java % ccat Demo.java
zsh: command not found: ccat
vedika.agrawal@FK1089-215056L Project_Java % cat Demo.java
Class Demo {
        public static void main(String args[])
        {
                System.out.println("This is my first java program");
        }
}
vedika.agrawal@FK1089-215056L Project_Java % git init
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:   git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint:   git branch -m <name>
Initialized empty Git repository in /Applications/Project_Java/.git/
vedika.agrawal@FK1089-215056L Project_Java % ls -a
.               ..              .git            Demo.java
vedika.agrawal@FK1089-215056L Project_Java %
```

Odin Tutorial for anyone who wants basics of git

Git commands
GitHub token passkey -
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

git —version - shows the version
pwd - current working directory
Touch Demo.java - to create a file
ls -a - to list all files + hidden files
Cat filename - to view the content of the file
Git init - initialise a git project
Git add . - to add all the files within the directory to the staging area
Git show - show the current status
Git commit -m "first commit" - to commit the staged files
Git remote add origin *repolink* - to connect to the remote repo
Git remote -v - to check the origin

Assignment
Create a project-UI/UX with the below structure
- Js(demo.js)
- css (demo.css)
- resource(demo.png)
- root(index.html)

Note - use git lifecycle commands for managing the UI/UX project and final push inside the relevant repository

Java (all files part of src)
.java
Source file
.class
Byte code->run on system

Java Project Creation

1. Packages (all in lowercase)
   Semantics -> com.flipkart.test
2. Classes (all in uppercase)
   Semantics -> Demo.Java
3. Properties (all in lowercase)
   Semantics -> .properties

Xyz.Java -> (compile Javac) -> xyz.class -> JVM -> Runtime Environment -> output

```
Important pointer about java 17 :-->

1. Java is OOP Language which support all the OO principles
2. In java , we use to create the class's / Object / package / properties / methods
3. In Java , There are various kind of class as we impelement during the project Impl .

      3.1 Bean Class:-->  which contains all the properties & respective setters / getters
      3.2 Business Class (Service Class):-- these class's contains the business logic related to the domain business
                           methods implementation.
      3.3 DAO class(Data access Object) :-- Generally these class's used to manage the database intraction Logic
      3.4 Other classs :-- Helper class's/ Validation class's / Utils Class's etc.
```

```
4. In Java Imple we will dicuss the following module for POS Flipfit Development :-->

    4.1 Class/ Object/ Constructor
    4.2 Array & properties
    4.3 Collection & Generics
    4.4 Exception Handling
    4.5 JDBC Impl
    4.6 I/O  input / Output Process
    4.7 New feature in JDK 17

            4.7.1 Date & Time API
            4.7.2 New Stream API impl in JDK 17
            4.7.3 Lambda Expression
            4.7.4 Default method in interface
            4.7.5 Logger Impl :-- apcahe logger

5. Creating Project Professional Doc using java tools & monitor the Java Application
```

Bean - data classes
business/service - logic classes
Client - main method will be here

```
Assignment :1 :-> create the SKT of Java POS flipfit APP with name:---  JEDI-FLIPFIT-POS-JAVA & define the
following structure of the package with classs :-->

    com.flipkart.bean  :-- based on the flipfit FR define all the possible bean class
    com.flipkart.business :-- same service class
    com.flipkart.client  :-- Client class
    com.flipkart.dao
    com.flipkart.utils
    com.flipkart.validator
    com.flipkart.exception
    com.flipkart.constant
```

Assignment 2 :-- In the Respective REPO :-->

JEDI-FLIPKART- GROUP-E-FLIPFIT

1. JEDI-FLIPFIT-POS-JAVA :-- complete push of Assign 1 :-->

2. JEDI-FLIPFIT-UML-ARTIFACTS :--:--- 1. USE CASE 2. ACTIVITY 3. CLASS-DIAGRAM 4. PROJECT-FR-CONCISE