Parking Lot Management System

Description

Design a Parking Lot Management System which has the capability to issue parking tickets, allocate and manage parking slots.

Features

1. Parking has 1 or more Entrances and Exits.
2. Each Parking has 1 or more Parking Slots
3. Parking slots are of 3 different types
    1. Cars
    2. Bikes
    3. Bicycles
4. There can be 0 or more Parking Slots of each type
5. When a vehicle comes at the entrance, the following scenarios need to be handled.
    1. No slot is available and the vehicle is not allowed inside the parking lot
    2. A slot is available and the customer decides to use it. The customer enters the parking lot
        1. Available Slot quantity is updated appropriately
6. Available Slot Selection can be governed by one of the following policies.
    1. Sequential
    2. Random Available
    3. More policies can be added later[**BONUS**]
7. When a vehicle comes at the Exit, following scenarios need to be handled
    1. Available Slot quantity is updated

Requirements

1. Onboard a new Parking Lot with a list of Parking Slots with type and capacity.
2. Issue Parking Tickets to vehicles entering the Parking Lot
3. The following system view should be available
    1. Available slots of each type
    2. Vehicles-In per Vehicle Type per Parking Entrance [**BONUS**]
    3. Vehicles-Out per Vehicle Type per Parking Exit[**BONUS**]

Other Notes

1. Do not use any database or NoSQL store, use in-memory data-structure for now.
2. Do not create any UI for the application.
3. Write a driver class/test cases for the demo purpose.
4. Please prioritize code compilation, execution and completion.
5. Work on the expected output first and then add good-to-have features of your own.

Expectations

1. Make sure that you have a working and demonstrable code
2. Make sure that code is functionally correct
3. **Make sure concurrent requests are handled appropriately**
4. Code should be modular and readable
5. Separation of concern should be addressed
6. Code should easily accommodate new requirements with minimal changes

7. Code should be easily testable

Test cases:

The strings in the test cases are indicative. Feel free to take exact parameters into your methods.

1. add_parking_lot(“PVR Koramangala”, “[1 Car Parkings, 1 Bike Parking, 1 Bicycle Parking]”, “2 Entry Gates”, “2 Exit Gates”)
2. is_available(“PVR Koramangala”, “Bike”) -> True (queried from Entry Gate 1)
3. is_available(“PVR Koramangala”, “Car”) -> True
4. is_available(“PVR Koramangala”, “Bike”) -> True (queried from Entry Gate 2)
5. park_vehicle(“PVR Koramangala”, “Car”, “Entry Gate 1”) -> True
6. park_vehicle(“PVR Koramangala”, “Bike”, “Entry Gate 2”) -> True
7. park_vehicle(“PVR Koramangala”, “Bike”, “Entry Gate 1”) -> False (This will fail because the spot was taken in the previous step)
8. is_available(“PVR Koramangala”, “Car”) -> False
9. print_all_available_slots(“PVR Koramangala”)

Total Available:	1
Car	0
Bike	0
Bicycle	1

10. print_total_in(“PVR Koramangala”, “Entry Gate 1”)

Car	1
Bike	0
Bicycle	0

11. print_total_out(“PVR Koramangala”, “Exit Gate 1”)

Car	0
Bike	0
Bicycle	0

12. unpark_vehicle(“PVR Koramangala”, “Bike”, “Exit Gate 1”)
13. print_total_out(“PVR Koramangala”, “Exit Gate 1”)

Car	0
Bike	1
Bicycle	0

FlipFit - SDE1

Design a backend system for a new enterprise application that Flipkart is launching, FlipFit.
Flipkart is partnering up with gyms across Bangalore to enter into the fitness space. For the Beta launch the requirements are as follows:

- There are only 2 centers for now - **Koramangala** and **Bellandur**. We might expand to multiple others if we get traction.
- Each center has 6 slots. 3 in the morning of **an hour each** from **6am to 9am** and similarly 3 in the evening from **6pm to 9pm.** The centers are open **7 days a week.**
- Each slot at a center can have only 2 possible workout variations for now - **Weights** and **Cardio.**
- The number of people that can attend each workout at each slot for a given station is fixed.
- User can perform the following operations:
    1. Register onto the platform
    2. View the workouts for a particular day
    3. Book a workout for a user if seats are available in that time slot at that center
    4. View his/her plan basis day as input

For simplicity's sake you can assume that the workout info will be entered by the Admin only once.
***Bonus** : Build an Admin view as well to modify the workout info at a center/slot level.*

Guidelines:

- Time: 90mins
- Write modular and clean code.
- **A driver program/main class/test case is needed to test out the code by the evaluator with multiple test cases**. But do not spend too much time in the input parsing. Keep it as simple as possible
- The evaluation criteria will be code functionality, modularity and extensibility
- You are not allowed to use any external databases like MySQL. Use only in memory data structures.
- No need to create any UX.

FlipFit - SDE2

Design a backend system for a new enterprise application that Flipkart is launching, FlipFit.
Flipkart is partnering up with gyms across Bangalore to enter into the fitness space. For the Beta launch the requirements are as follows:

- There are only 2 centers for now - **Koramangala** and **Bellandur**. We plan to expand to multiple others if we get traction.
- Each center has **n slots of an hour each**. For eg the Bellandur center has only 6 slots - 3 in the morning of **an hour each** from **6am to 9am** and similarly 3 in the evening from **6pm to 9pm.** All the centers are open **7 days a week.**
- Each slot at a center can have **n possible workout variations** - **Weights**, **Cardio, Yoga, Swimming etc.** There could be newer workouts added at center/slot level in the future. Adding new workout types should be easy and lead to minimal changes in application.
- The number of seats in each workout at each time-slot for a given center is fixed.
- For simplicity's sake you can assume that the workout info will be entered by the Admin only once.
- View his/her plan basis day/workout type/center as input. It should be easy to change view pattern.
- User can perform the following operations:
    - Register onto the platform
    - View the workouts availability/unavailability for a particular day
    - Book a workout for a user if seats are available in that time slot at that center

Bonus: return the nearest time slot for the same workout/center/user combination, keeping in mind his other booked slots

Guidelines:

- Time: 90mins
- Write modular and clean code.
- **A driver program/main class/test case is needed to test out the code by the evaluator with multiple test cases**. But do not spend too much time in the input parsing. Keep it as simple as possible.
- Evaluation criteria: Demoable & functionally correct code, Code readability, Proper Entity modelling, Modularity & Extensibility, Separation of concerns, Abstractions. Use design patterns wherever applicable
- You are not allowed to use any external databases like MySQL. Use only in memory data structures.
- No need to create any UX
- Please focus on the Bonus Feature only after ensuring the required features are complete and demoable.

FlipFit - SDE3

Design a backend system for a new enterprise application that Flipkart is launching, FlipFit.
Flipkart is partnering up with gyms across Bangalore to enter into the fitness space. For the Beta launch the requirements are as follows:

- There are multiple centers in Bangalore. We can expand to multiple cities across and beyond India.
- Each center has **n slots of an hour each**. For eg the Bellandur center has only 6 slots - 3 in the morning of **an hour each** from **6am to 9am** and similarly 3 in the evening from **6pm to 9pm.** All the centers might/might not be open **7 days a week.**
- Each slot at a center can have **n possible workout variations** - **Weights**, **Cardio, Yoga, Swimming etc.** There could be newer workouts added at center/slot level in the future.
- The number of seats in each workout at each time-slot for a given center is fixed.
- Handle concurrent blocking of workout slots and ensure no overbooking or double booking of slots
- It should be easy to change the underlying data storage mechanism as data scale increases without changing various parts of the application
- For simplicity's sake you can assume that the workout info will be entered by the Admin only once.
- User can perform the following operations:
    - Register onto the platform
    - View the workouts availability/unavailability for a particular day
    - Book a workout for a user if seats are available in that time slot at that center
    - View his/her plan basis day as input
    - Cancel his/her workout

Bonus:

- *Return the nearest time slot for the same workout/center/user combination, keeping in mind his other booked slots*
- *Once the seats get filled up, there is a waiting list with a fixed number of trainees. Also notify waitlisted candidates that he/she has been promoted*

Guidelines:

- Time: 90mins
- Write modular and clean code.
- **A driver program/main class/test case is needed to test out the code by the evaluator with multiple test cases**. But do not spend too much time in the input parsing. Keep it as simple as possible.
- Please handle concurrency wherever applicable.
- Evaluation criteria: Demoable & functionally correct code, Code readability, Proper Entity modelling, Modularity & Extensibility, Separation of concerns, Abstractions. Use design patterns wherever applicable
- You are not allowed to use any external databases like MySQL. Use only in memory data structures.
- No need to create any UX
- Please focus on the Bonus Feature only after ensuring the required features are complete and demoable.

ONLY FOR INTERVIEWERS SECTION

SDE2 Evaluation - Some Points to note:

1. Entities: User, Center, Workout, Slot,
    Workout-Slot-User association(inventory/seats entity)
    Booking/Order entity
    Separate entities for Views is good.
2. Separation of concerns: DAO layer for entity CRUD operations. If not implemented we can ask during eval on whether they come up with a DAO layer. Also can ask why a DAO layer is needed in the 1st place.
3. Extensibility: Adding more centres, workouts, slots, different views easily.
4. Design pattern: View logic/criteria pluggable. Can expect Strategy. If not implemented we can ask during eval on how they'd implement.
5. Exception & corner case handling.
6. Concurrency:

Should theoretical knowledge of concurrency handling be a bonus or an expectation?

    During evaluation, we can ask:
    1. Identify Critical sections in the code where concurrency issues would arise
    2. Identify non thread-safe parts of code like Map vs ConcurrentHashMap & about Optimistic Locking etc.
7. Code should be modular & unit-testable.

We can scale up/down the above for SDE3/SDE1.

Job Portal Application (SDE1)

Build a mini job portal application that enables users to search for people, companies or job-openings. The portal would be supporting records containing the below attributes: • Name

- Category
- Company Name
- Designation
- Experience
- Skillset
- Location

E.g.

{ “name” : “Sherlock Holmes”, “category”: “people”, “company” : “Baker Street Detective Solutions”, “designation” : “senior detective”, “experience” : 10, “skillset”: “smart”, “location”: “Baker Street” }

{ “name” : “opening-1”, “category”: “opening”, “company” : “Baker Street Detective Solutions”, “designation”: “assistant detective”, “experience” : 5, “skillset”: “smart”, “location” : “Baskerville” }

Requirements

The application should be able to support the below requirements

1. Add a record
2. Modify/delete already added record
3. Search for entities containing either a single word or a collection of words,e.g. - searching for “detective”, should return results containing the term detective in their document
   - searching for “smart detective”, should return results containing either “smart” or “detective” or both
4. Return results for a given attribute being equal to a given value, e.g. searching for “location=baker street” should return results whose location is “baker street”

Bonus: sort/order the results based on a specific attribute

Guidelines

- Documents should be stored in memory, use of any external DB is not allowed. Use of external libraries for searching and indexing is NOT permitted however you
- are free to use all possible constructs of your favourite language.

Evaluation Criteria

Separation of concerns, Abstraction, Testability, Code readability, Language proficiency, Efficiency