# Question 1:

1. **Class Booking:**

**Attributes** - bookingID: int [1] - startDate: string [1] - duration: string [1] - status: string [1] = "on-hold" - userID: int [1] - resourceID: int [1] - occupantsNum: int [0..1] - transactionID: int [0..1] - sqlHelper BookingSqlHelper[1]

**Operations** + retriveID() : int + retriveStartDate() : String + retriveDuration() : int + retriveStatus(String status) : String + retriveResource() : Resource + confirm() : void + cancel() : boolean + archive() : void

**Class BookingAPI_Impl:**

- activeBookings : BookingCollection [1] - equipment : EquipmentCollection [1] - rooms : RoomCollection [1]

+ addBooking(startDate: string, duration: int) : void + cancelBooking(bookingID: int): boolean + checkResourceAvail(ID: int)

**Class Room:**

- roomID: int [1] - roomName: string [1] - capacity: int [1] - isAvail: boolean [1] = true status: string [1] = "avaliable" - sqlHelper: RoomSqlHelper [1]

+ setStatus(status: string): void + getStatus(): string + getName(): string + checkAvail(): boolean + getCapacity(): int

**Class Equipment:**

- equipmentID: int [1] - equipmentType: string [1] - depositAmount: int [1] - status: int [1] = "avaliable" `- sqlHelper: EquipmentSqlHelper [1]

+ getID() : int + getEquipmentType() : string + getDepositAmount() : int + getStatus() : string + setStatus(status : string) : void #### Class **StorageAPI_Impl**:

- bookingSqlHelper: BookingSqlHelper - resourceSqlHelper: ResourceSqlHelper

+ saveBooking(booking: Booking): boolean + updateBooking(booking: Booking): boolean + deleteBooking(bookingID: String): boolean

2.

**`storageAPI -- StorageAPI_Impl`** Relationship: Implementation Multiplicity: N/A Navigability: `StorageAPI_Impl` to `StorageAPI`

**`BookingAPI_Impl -- BookingAPI`** Relationship: Implementation Multiplicity: N/A Navigability: `BookingAPI_Impl` to `BookingAPI`

**`BookingAPI_Impl -- BookingCollection`** Relationship: Composition Multiplicity: 1 : 1 Navigability: `BookingAPI_Impl` to `BookingAPI`

**`BookingAPI_Impl -- StorageAPI_Impl`** Relationship: Aggregation Multiplicity: 1 to 1 Navigability: `BookingAPI_Impl` to `StorageAPI_Impl`

**`Resource -- Equipment`** Relationship: Inheritance Multiplicity: N/A Navigability: N/A

**`BookingCollection -- Booking`** Relationship: Aggregation Multiplicity: 1 : 0..* Navigability: `BookingCollection` to `Booking`

**`Booking -- StorageAPI`** Relationship: Aggregation Multiplicity: N/A Navigability: `Booking` to `StorageAPI`

**`BookingSqlHelper -- SQLHelper`** Relationship: Inheritance Navigability: `BookingSqlHelper` to `SQLHelper`

3. `sql` has instance scope. This is because each instance of an SQL Helper class must have its own unique query string, therefore `sql` is directly attached to a specific instance of a class.

4. Resource is an abstract class. This is because Resource represents a general concept of a Room or Equipment in the CBS. It acts as a base class for other classes that will provide the implementation of manipulating resources. Resource requires a constructor, as it has attributes that should be initialised such as `ID, name, type`.

5. The attributes `ID, name, type, string` in `Resource` must be kept private. This is important to maintain encapsulation within the class and avoid unexpected behaviour. All operations should remain public, so they can be accessed by other classes such as `BookingAPI_Impl`. Any class attributes that need to be accessed can do so through getter and setter elements.

6. This is necessary because the return of equipment is triggered by the staff who need to authorise the equipment return. The booking is active until that point, so this is needed for termination of a booking, therefore it must also release the deposit if applicable.

# Question 3

![[image-110.png]]

# Question 4

![[image-112.png]]

# Question 5

### Test 1: MakeABooking - Main Flow

![[image-115.png]] ![[image-116.png|581x455]]

### Test 2: MakeABooking: InvalidBookingRequest - Alternative Flow

![[image-117.png]]![[image-119.png]]