

**[JAVA Hotel SYSTEM]**

**Assignment Report**



**STUDENT ID: K00223470**

**STUDENT: MUHAMMAD MURTAZA**

**Room.java**The idea is that Room class contains a collection of references (String) and each reference has collection of days (HashSet<Integer>). There is direct relationship between a booking reference and set of days. There is 1-to-many relationship which is achieved by the HashMap<String, HashSet<Integer>> type object bookingRefDaysMapper. This design allows us to have several bookings with number of days for a room class. The diagram below depicts this idea.



All the methods of Room class benefited with constant time lookup and access for booking reference that is of HashMap<String, HashSet<Integer>> type. Here is the summary of each method:

1. **void book(String bookingRef, Integer[] days:** It serves 2 responsibilities. It adds a new booking and updates an existing one. There is not validation for booking reference name and the days conflict because that’s dealt in the Hotel.java class. Semaphore is used to synchronise the access to HashMap<String, HashSet<Integer>>.
2. **HashSet<Integer> getAllBookedDays():** Loops through each booking and its days to populate a HashSet<Integer> type hashset with the days and returns back to the calling code.
3. **HashSet<Integer> getAllBookedDays(String skipBookingRef):** This is an overload that allows us to get all the booked days except for the given name. This helps us when updating an existing booking because we do not want to create the conflict between the booking that is being updated with itself.
4. **void cancelBooking(String bookingRef) throws NoSuchBookingException:**

Removes the given booking reference from HashMap<String, HashSet<Integer>>. Throws exception if the booking reference does not exist. Here again Semaphore is used to synchronise the access to HashMap<String, HashSet<Integer>>. This is to prevent double deletion of the same booking refence.

**Hotel.java**This class can be treated as a “wrapper” around the Room class, its responsibility is to create rooms and bookings and only remove existing bookings. A booking consists of room(s) and the day(s). Hotel class also works as a security layer for the room class. No Room instance / object is accessed by the clients of Hotel class. Following diagram depicts this concept:



There are 2 HashMaps, one to map room number with room’s instance (1-to-1 mapping) - HashMap<Integer, Room2> roomMap, while the other one maps booking reference to the list of room numbers (1-to-many) HashMap<String, List<Integer>> bookingRefRoomNumMap. The roomMap is used to get constant time access to the room instance for given room number, and bookingRefRoomNumMap manages the booking reference for one or more than one room – this is to support the extended specification that requires multiple rooms booking / update and cancelation using single booking reference. The same booking refence can exist in multiple rooms when boolean bookRooms(String bookingRef, Integer[] days, int[] roomNums) throws NoSuchBookingException method is used. The same booking reference is used for the individual rooms as well as for bookingRefRoomNumMap object which keeps the mapping between the booking refence and room(s). Semaphore is used to thread-safe the code in bookRoom, bookRooms, updateBooking (and its overload) & cancelBooking methods. The reason is to protect dirty reads/writes in the underlying data structures. ConcurrentHashMap<K,V> could also be used to make the code thread-safe but I found that late and due to time constraint stick to the normal HashMap.

1. **boolean roomBooked(Integer[] days, int roomNumber):** Gets the existing booked days for given roomNum – it uses roomMap - HashMap for the constant time lookup. Once the days are retrieved, it tries to find any matching day in linear time and return true at the first match found. Returns false when no match is found.
2. **boolean bookRoom(String bookingRef, Integer[] days, int roomNum):** It acquires the Semaphore lock (since the data is being saved / added) and validates if the days have no conflict with the existing booking. It adds the new booking reference along with its days in the room object and saves the booking reference and room number map in bookingRefRoomNumMap.
3. **boolean updateBooking(String bookingRef, Integer[] days, int roomNum) throws NoSuchBookingException**: It throws NoSuchBookingException exception if invalid booking refence is given, acquires the Semaphore lock. Uses roomMap to retrieve the requested room object (again constant time lookup). A helper method CanUpdate is used here to determine if there is any conflict with any other bookings in given room number. If not updates the booking. Its overload method loops through each calls this method.

1. void CancelBooking(String bookingRef) throws NoSuchBookingException: it throws NoSuchBookingException exception if invalid booking refence is given. Removes the booking reference from the room(s) and its entry from bookingRefRoomNumMap object. I wish I could have made these 2 removals in an ACID like transaction – Atomicity, Consistency, Isolation, Durability.
2. **boolean canUpdate(String bookingRef, Integer[] days, int roomNum)**: Determines if its safe to update any booking. It checks for any conflicts between new days and already saved booking’s days except the one which is being updated.
3. **boolean bookRooms(String bookingRef, Integer[] days, int[] roomNums) throws NoSuchBookingException**: Checks if the bookingRef already exists, in which case throws exception because that should be an update request not a new one. After acquiring the lock, it uses roomsBooked method to check if there is not conflict, once pass that, it loops through each room and books it is using the same booking reference for all room for the same days. The same process is reversed when cancelling the bookings. Finally, it updates bookingRefRoomNumMap with new booking reference and associated room numbers.
4. **boolean roomsBooked(Integer[] days, int[] roomNums):** This method is nothing but a wrapper method for roomBooked. It loops through each room number and calls roomBooked to check for the conflict. Returns true as soon as it finds the conflict, rest of the room numbers are not checked.