Design a Hotel Management System

Let's design a hotel management system.

We'll cover the following

A Hotel Management System is a software built to handle all online hotel activities easily and safely. This System will give the hotel management power and flexibility to manage the entire system from a single online portal. The system allows the manager to keep track of all the available rooms in the system as well as to book rooms and generate bills.

System Requirements

We'll focus on the following set of requirements while designing the Hotel Management System:

1. The system should support the booking of different room types like standard, deluxe, family suite, etc.
2. Guests should be able to search the room inventory and book any available room.
3. The system should be able to retrieve information, such as who booked a particular room, or what rooms were booked by a specific customer.
4. The system should allow customers to cancel their booking - and provide them with a full refund if the cancelation occurs before 24 hours of the check-in date.
5. The system should be able to send notifications whenever the booking is nearing the check-in or check-out date.
6. The system should maintain a room housekeeping log to keep track of all housekeeping tasks.
7. Any customer should be able to add room services and food items.
8. Customers can ask for different amenities.
9. The customers should be able to pay their bills through credit card, check or cash.

Use case diagram

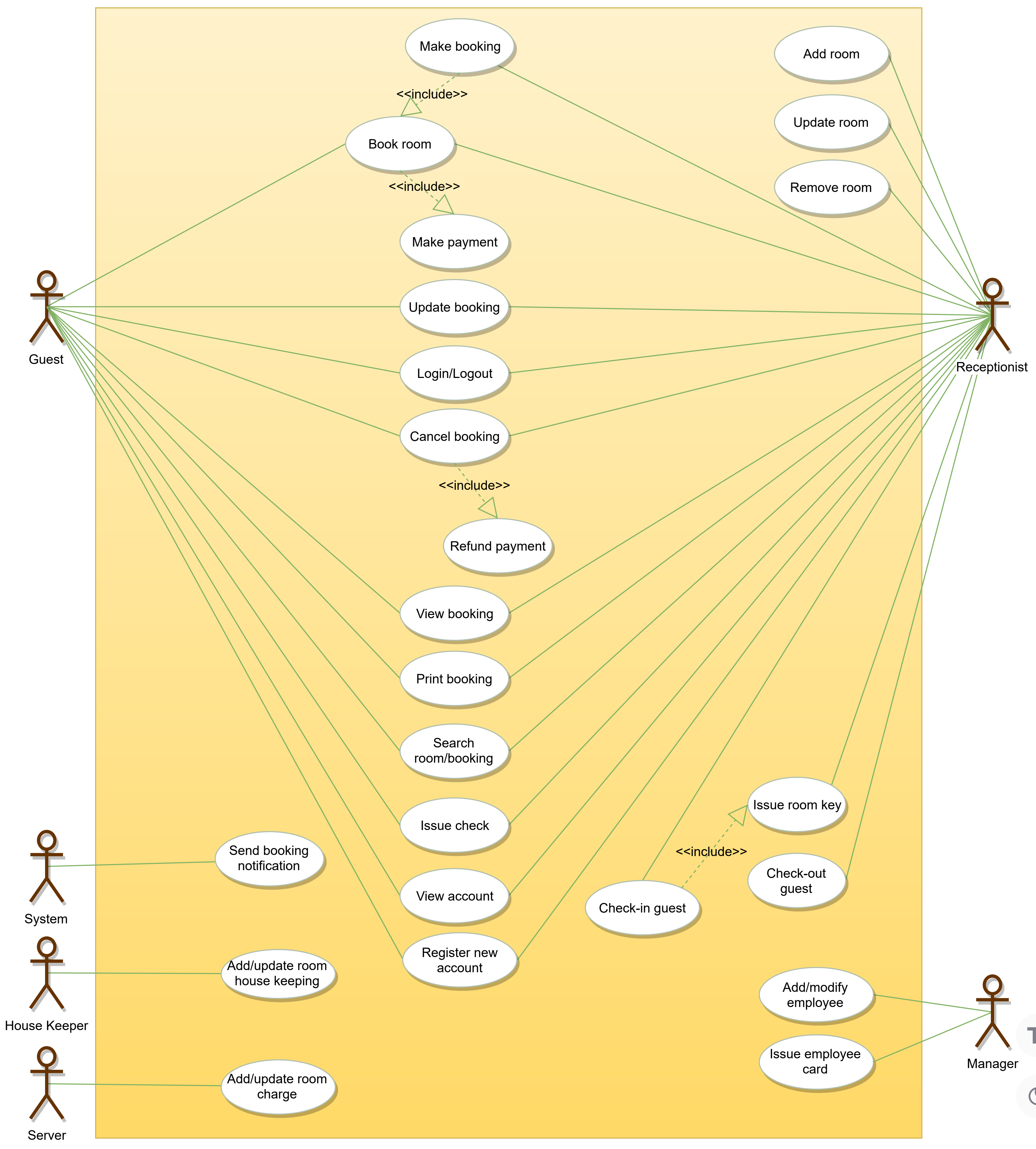
Here are the main Actors in our system:

- **Guest:** All guests can search the available rooms, as well as make a booking.

- **Receptionist:** Mainly responsible for adding and modifying rooms, creating room bookings, check-in, and check-out customers.
- **System:** Mainly responsible for sending notifications for room booking, cancellation, etc.
- **Manager:** Mainly responsible for adding new workers.
- **Housekeeper:** To add/modify housekeeping record of rooms.
- **Server:** To add/modify room service record of rooms.

Here are the top use cases of the Hotel Management System:

- **Add/Remove/Edit room:** To add, remove, or modify a room in the system.
- **Search room:** To search for rooms by type and availability.
- **Register or cancel an account:** To add a new member or cancel the membership of an existing member.
- **Book room:** To book a room.
- **Check-in:** To let the guest check-in for their booking.
- **Check-out:** To track the end of the booking and the return of the room keys.
- **Add room charge:** To add a room service charge to the customer's bill.
- **Update housekeeping log:** To add or update the housekeeping entry of a room.

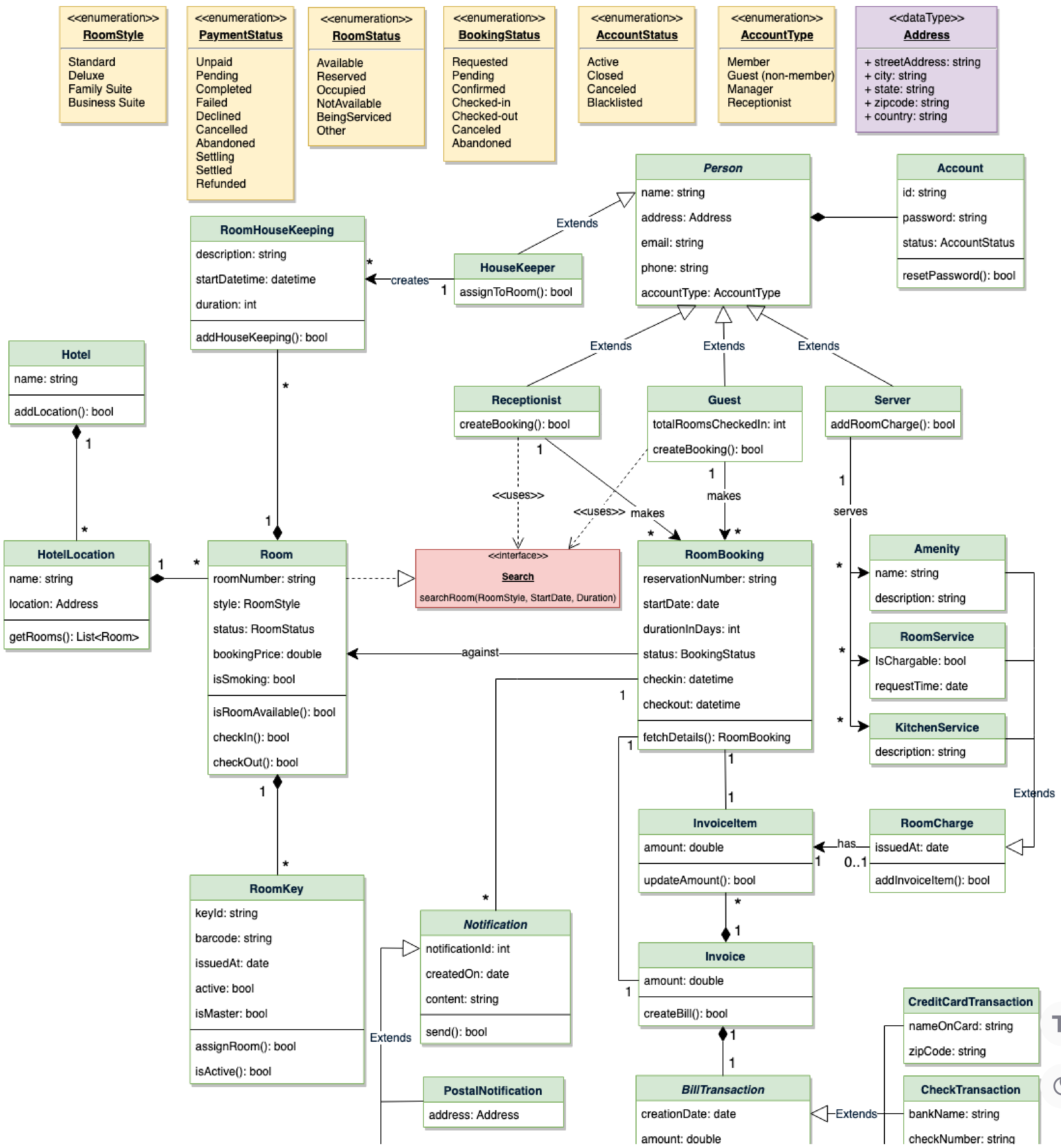


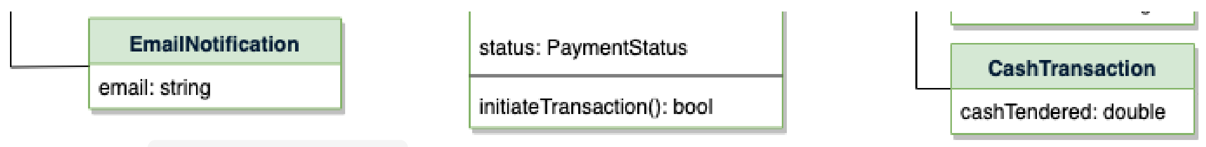
Use case diagram

Class diagram

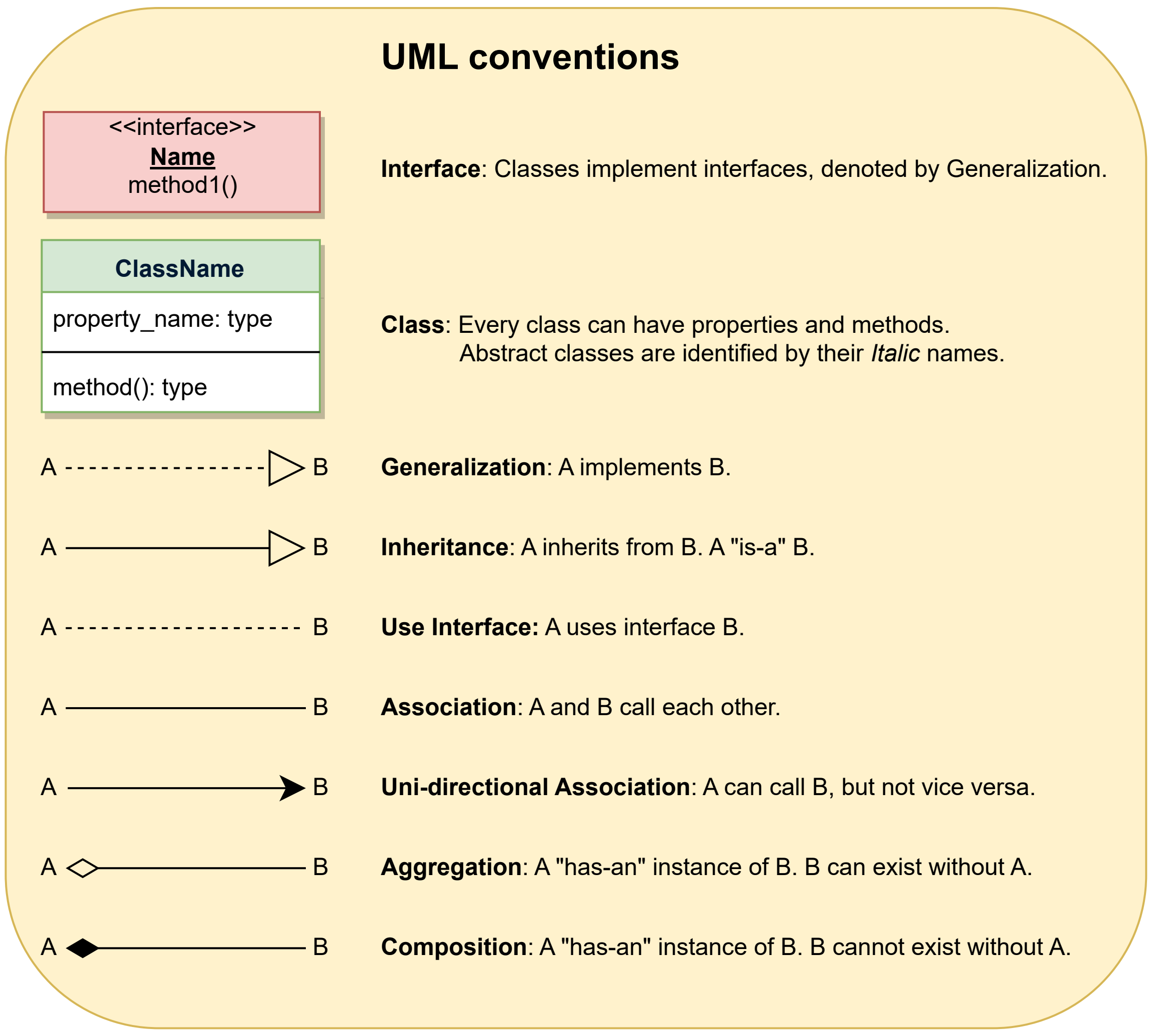
Here are the main classes of our Hotel Management System:

- **Hotel and HotelLocation:** Our system will support multiple locations of a hotel.
- **Room:** The basic building block of the system. Every room will be uniquely identified by the room number. Each Room will have attributes like Room Style, Booking Price, etc.
- **Account:** We will have different types of accounts in the system: one will be a guest to search and book rooms, another will be a receptionist. Housekeeping will keep track of the housekeeping records of a room, and a Server will handle room service.
- **RoomBooking:** This class will be responsible for managing bookings for a room.
- **Notification:** Will take care of sending notifications to guests.
- **RoomHouseKeeping:** To keep track of all housekeeping records for rooms.
- **RoomCharge:** Encapsulates the details about different types of room services that guests have requested.
- **Invoice:** Contains different invoice-items for every charge against the room.
- **RoomKey:** Each room can be assigned an electronic key card. Keys will have a barcode and will be uniquely identified by a key-ID.



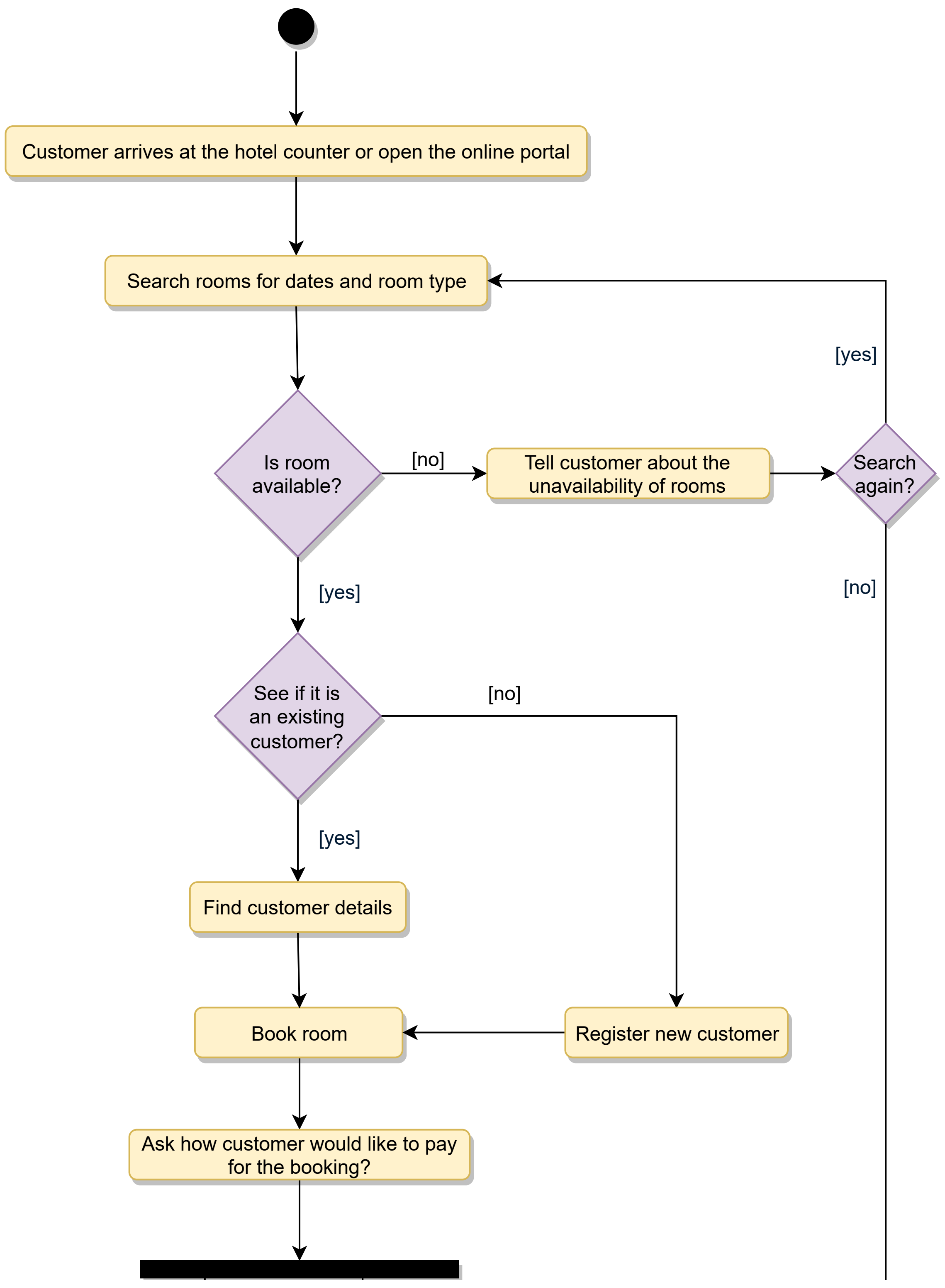


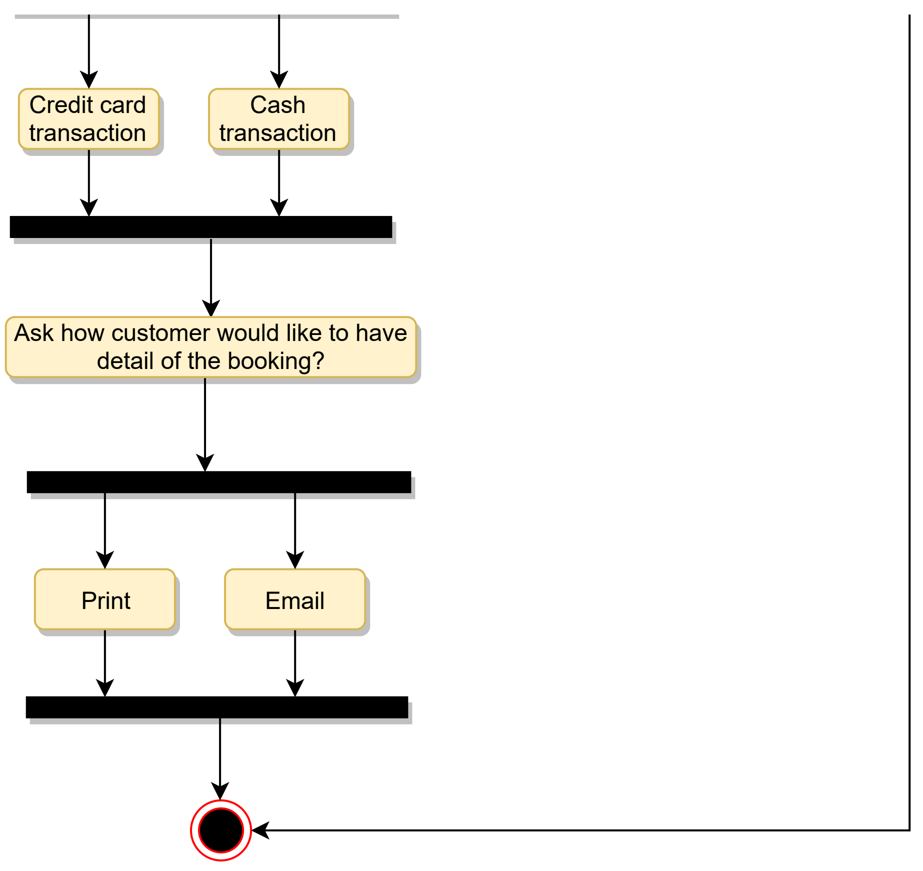
Class diagram


UML conventions
<<interface>>
Name
method1()
Interface: Classes implement interfaces, denoted by Generalization.
ClassName
property_name: type
method(): type
Class: Every class can have properties and methods. Abstract classes are identified by their Italic names.
A B
Generalization: A implements B.
A B
Inheritance: A inherits from B. A "is-a" B.
A B
Use Interface: A uses interface B.
A B
Association: A and B call each other.
A B
Uni-directional Association: A can call B, but not vice versa.
A B
Aggregation: A "has-an" instance of B. B can exist without A.
A B
Composition: A "has-an" instance of B. B cannot exist without A.

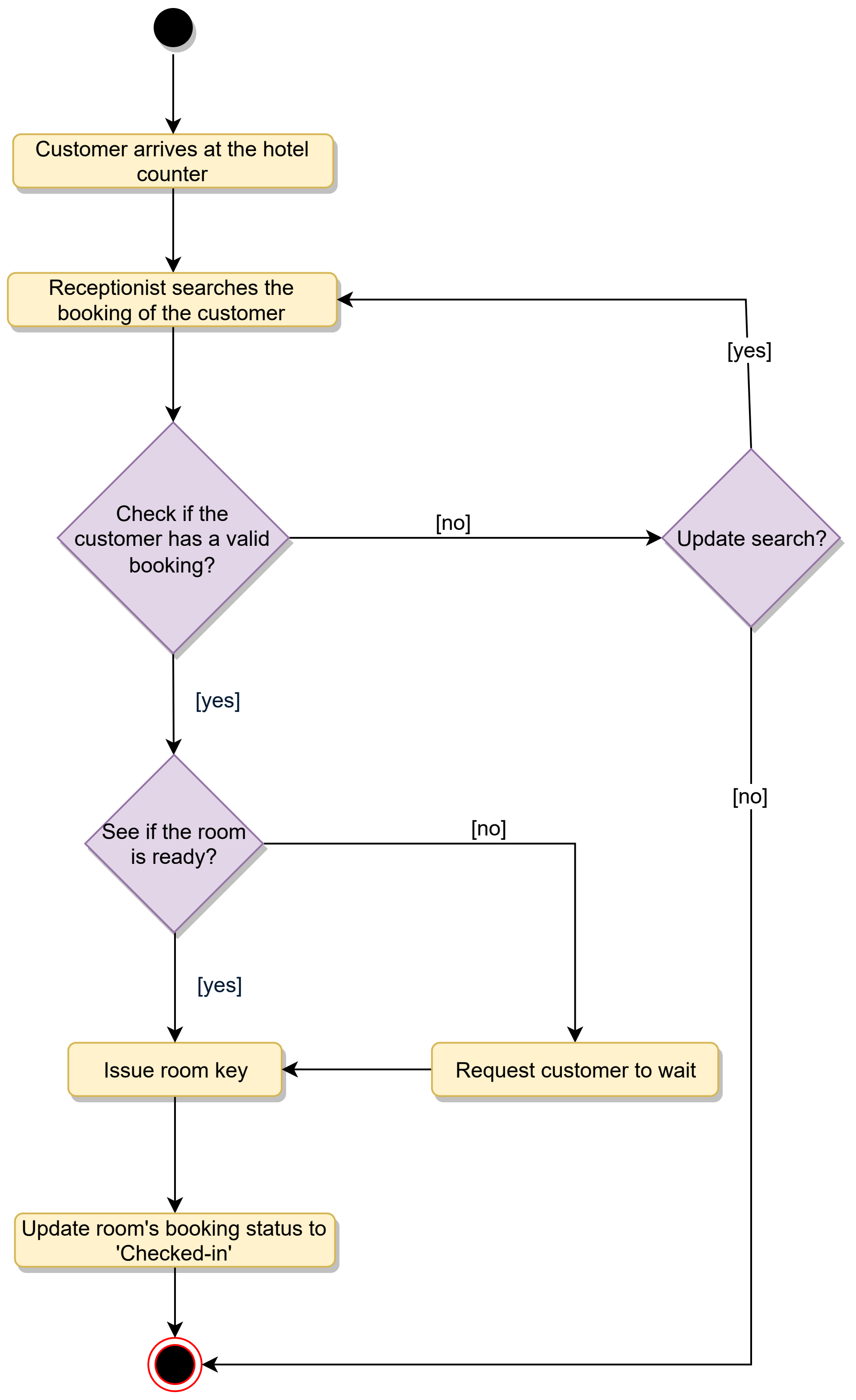
Activity diagrams

Make a room booking: Any guest or receptionist can perform this activity. Here are the set of steps to book a room:

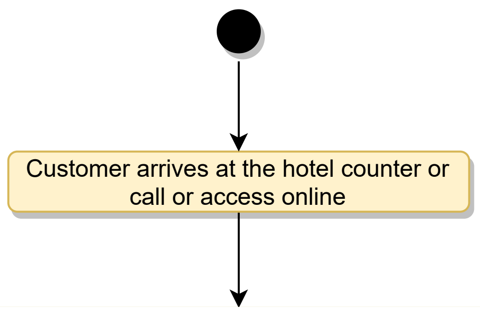

Customer arrives at the hotel counter or open the online portal
Search rooms for dates and room type
[yes]
Is room available?
[no]
Tell customer about the unavailability of rooms
Search again?
[yes]
[no]
See if it is an existing customer?
[no]
[yes]
Find customer details
Book room
Register new customer
Ask how customer would like to pay for the booking?

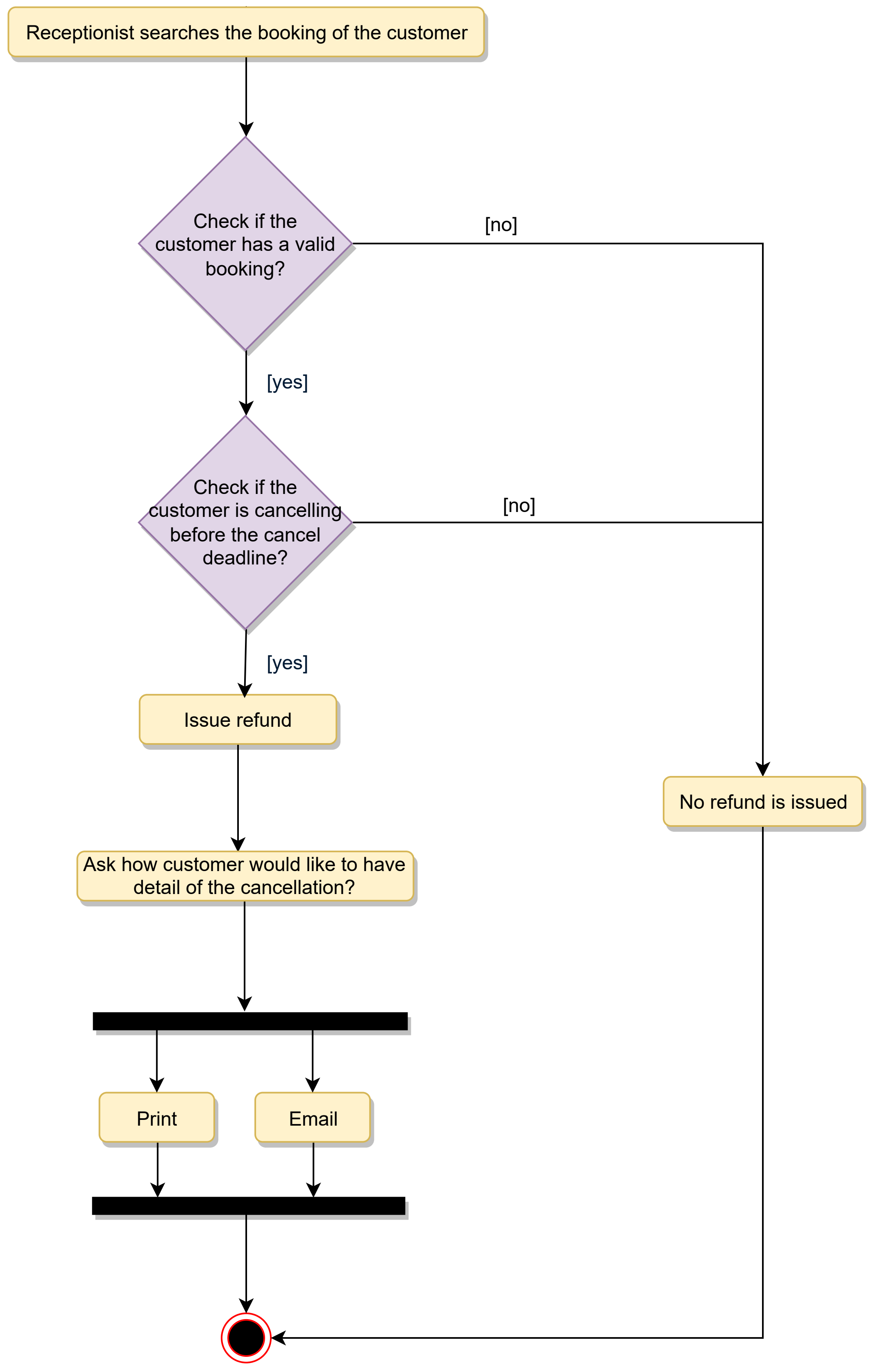


Check in: Guest will check in for their booking. The Receptionist can also perform this activity. Here are the steps:


Customer arrives at the hotel counter
Receptionist searches the booking of the customer
[yes]
Check if the customer has a valid booking?
[no]
Update search?
[yes]
[no]
See if the room is ready?
[no]
[yes]
Issue room key
Request customer to wait
Update room's booking status to 'Checked-in'

Cancel a booking: Guest can cancel their booking. Receptionist can perform this activity. Here are the different steps of this activity:


Customer arrives at the hotel counter or call or access online


Receptionist searches the booking of the customer
Check if the customer has a valid booking?
[no]
[yes]
Check if the customer is cancelling before the cancel deadline?
[no]
[yes]
Issue refund
No refund is issued
Ask how customer would like to have detail of the cancellation?
Print
Email

Code

Here is the high-level definition for the classes described above.

Enums, data types, and constants: Here are the required enums, data types, and constants:

Java Python

```
class RoomStyle(Enum):
  STANDARD, DELUXE, FAMILY_SUITE, BUSINESS_SUITE = 1, 2, 3, 4


class RoomStatus(Enum):
  AVAILABLE, RESERVED, OCCUPIED, NOT_AVAILABLE, BEING_SERVICED, OTHER = 1, 2, 3, 4, 5, 6


class BookingStatus(Enum):
  REQUESTED, PENDING, CONFIRMED, CHECKED_IN, CHECKED_OUT, CANCELLED, ABANDONED = 1, 2, 3, 4, 5, 6, 7


class AccountStatus(Enum):
  ACTIVE, CLOSED, CANCELED, BLACKLISTED, BLOCKED = 1, 2, 3, 4, 5


class AccountType(Enum):
  MEMBER, GUEST, MANAGER, RECEPTIONIST = 1, 2, 3, 4


class PaymentStatus(Enum):
  UNPAID, PENDING, COMPLETED, FILLED, DECLINED, CANCELLED, ABANDONED, SETTLING, SETTLED, REFUNDED = 1, 2, 3, 4,
```

```python
class Address:
  def __init__(self, street, city, state, zip_code, country):
    self.__street_address = street
```

Account, Person, Guest, Receptionist, and Server: These classes represent the different people that interact with our system:

Java Python

```python
# For simplicity, we are not defining getter and setter functions. The reader can
# assume that all class attributes are private and accessed through their respective
# public getter methods and modified only through their public methods function.

class Account:
  def __init__(self, id, password, status=AccountStatus.Active):
    self.__id = id
    self.__password = password
    self.__status = status

  def reset_password(self):
    None


# from abc import ABC, abstractmethod
class Person(ABC):
  def __init__(self, name, address, email, phone, account):
    self.__name = name
    self.__address = address
    self.__email = email
    self.__phone = phone
    self.__account = account


class Guest(Person):
  def __init__(self):
    self.__total_rooms_checked_in = 0
```

Hotel and HotelLocation: These classes represent the top-level classes of the system:

Java Python

```python
class HotelLocation:
  def __init__(self, name, address):
    self.__name = name
    self.__location = address

  def get_rooms(self):
    None


class Hotel:
  def __init__(self, name):
    self.__name = name
    self.__locations = []

  def add_location(self, location):
    None
```

Room, RoomKey, and RoomHouseKeeping: To encapsulate a room, room key, and housekeeping:

Java Python

```python
from abc import ABC, abstractmethod

class Search(ABC):
  def search(self, style, start_date, duration):
    None


class Room(Search):
  def __init__(self, room_number, room_style, status, price, is_smoking):
    self.__room_number = room_number
    self.__style = room_style
    self.__status = status
    self.__booking_price = price
    self.__is_smoking = is_smoking

    self.__keys = []
    self.__house_keeping_log = []

  def is_room_available(self):
    None

  def check_in(self):
    None

  def check_out(self):
    None
```

RoomBooking and RoomCharge: To encapsulate a booking and different charges against a booking:

Java | Python

```
    class RoomBooking:
      def __init__(self, reservation_number, start_date, duration_in_days, booking_status):
        self.__reservation_number = reservation_number
        self.__start_date = start_date
        self.__duration_in_days = duration_in_days
        self.__status = booking_status
        self.__checkin = None
        self.__checkout = None

        self.__guest_id = 0
        self.__room = None
        self.__invoice = None
        self.__notifications = []

      def fetch_details(self, reservation_number):
        None


    # from abc import ABC, abstractmethod
    class RoomCharge(ABC):
      def __init__(self):
        self.__issue_at = datetime.date.today()

      def add_invoice_item(self, invoice):
        None
```

← Back
Design Blackjack and a Deck of Cards

Next →
Design a Restaurant Management system

✓ Mark as Completed