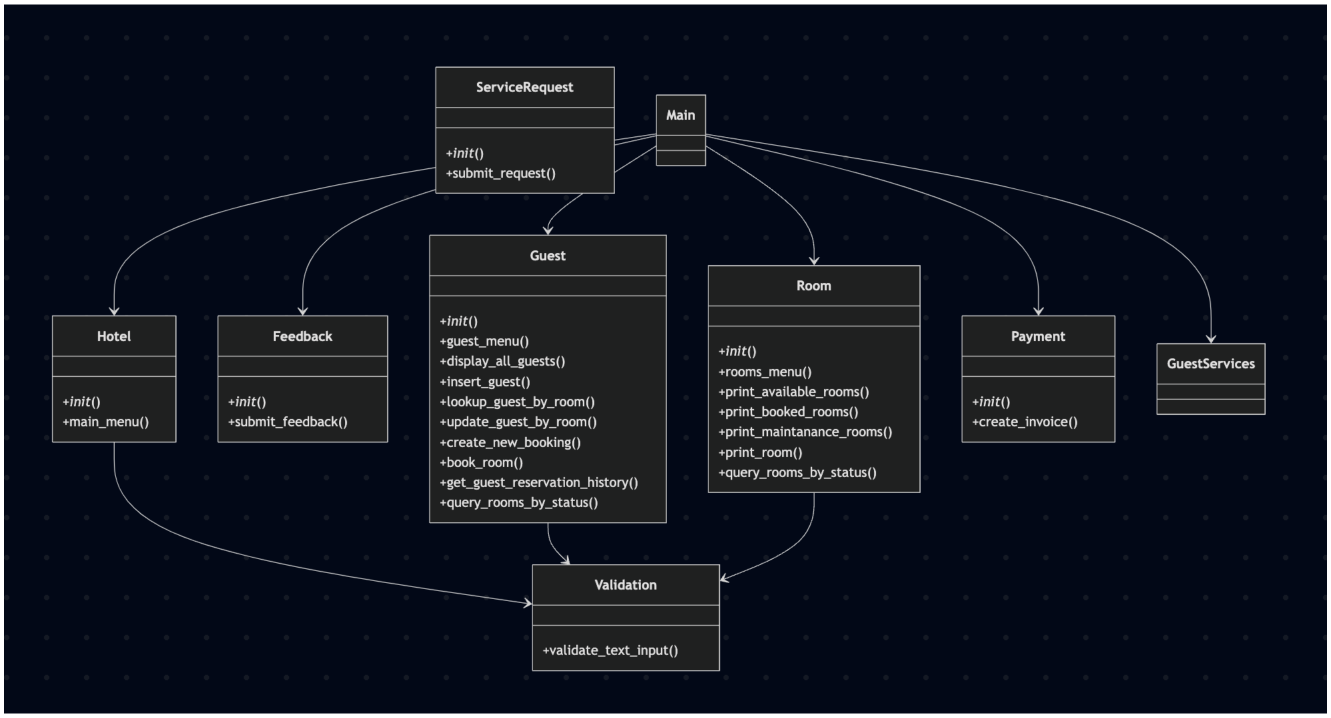
Royal Stay Hotel Management System

Abdulrahman Alzaabi

ICS220 Program. Fund.

Areej Abdulfattah

A. UML case diagram:



B. **Description**

The UML class diagram consists of seven classes:

- **Hotel**: Central class managing rooms, guests, and bookings.
- **Room**: Represents hotel rooms with availability functionality.
- **Guest**: Stores guest information and loyalty points.
- **Booking**: Links guests to rooms for specific dates.
- **Payment**: Manages payment details for bookings.
- **ServiceRequest**: Handles guest service requests.
- **Feedback**: Records guest feedback for bookings.

Relationships:

- **Composition**: Hotel owns Room, Guest, and Booking.
- **Association**: Booking connects to Guest, Room, and Payment; ServiceRequest to Guest; Feedback to Booking.

C.

Code:

```
import sqlite3
from datetime import date
from roomManagement import Room
from guestManagement import Guest
from bookingManagement import Booking
from paymentInvoicing import Payment
from guestServices import ServiceRequest as GuestServices
from feedbackReviews import Feedback
from inputValidation import Validation
from hotelSystem import Hotel

# Database file name
db_name = 'data.sqlite'

# Connect to the SQLite database (it will be created if it doesn't exist)
conn = sqlite3.connect(db_name)
cursor = conn.cursor()

# Initialize modules
Room = Room(conn=conn)
Guest = Guest(conn=conn)
Booking = Booking(conn=conn)
Payment = Payment(conn=conn)
GuestServices = GuestServices(conn=conn)
Feedback = Feedback(conn=conn)
Hotel = Hotel(Room=Room, Guest=Guest)




if __name__ == "__main__":
    ascii_art = "   ___  ___  ___   _____ ______   _______   ___  _________  _________  ___  ___      \n   |\\  \\ |\\  \\|\\  \\ |\\   _ \\
_   \\ |\\  ___\\  |\\  \\ |\\   __  \\ |\\   __  \\ |\\  \\|\\  \\    \n   \\ \\  \\\\ \\  \\\\\\  \\\\ \\  \\\\\\__\\ \\  \\\\ \\   __/| \\ \\  \\\\ \\  \\|\\  \\\\ \\  \\|\\  \\\\ \\
\\\\\\  \\   \n __ \\ \\  \\\\ \\  \\\\\\  \\\\ \\  \\\\|__| \\  \\\\ \\  \\_|/__\\ \\  \\\\ \\   _ _\\\\ \\   __  \\\\ \\   __  \\  \n|\\  \\\\_\\  \\\\ \\  \\\\\\  \\\\ \\  \\
\\ \\  \\\\ \\  \\_|\\ \\\\ \\  \\\\ \\  \\\\  \\|\\ \\  \\ \\  \\\\ \\  \\ \\  \\ \n\\ \\_________\\\\ \\_______\\\\ \\__\\    \\ \\__\\\\ \\________\\\\ \\__\\\\
\\__\\\\ _\\ \\ \\__\\ \\__\\\\ \\__\\ \\__\\\n \\|_________| \\|________| \\|__|     \\|__| \\|________| \\|__| \\|__|\\|__| \\|__|\\|__| \\|__|\\|__|\n
___  ___   _________   __________   ________   ___                                \n|\\  \\|\\  \\ |\\   __  \\ |\\____   ____\\|\\   ____ \\  |\\  \\
\n\\ \\  \\\\\\  \\\\ \\  \\|\\  \\\\|___ \\  \\_|\\ \\   __/| \\ \\  \\                          \n \\ \\  __  \\\\ \\  \\\\\\  \\   \\ \\  \\  \\ \\  \\_|/__\\ \\  \\
\n  \\ \\  \\ \\  \\\\ \\  \\\\\\  \\   \\ \\  \\  \\ \\  \\_|\\ \\\\ \\  \\____                   \n   \\ \\__\\ \\__\\\\ \\________\\    \\ \\__\\  \\ \\
\\________\\\\ \\_______\\                    \n    \\|__|\\|__| \\|_______|     \\|__|   \\|________| \\|________|                       \n"
    print(ascii_art)
    print("Jumeirah Hotel by Abood")
    print("Best Hotel Ever, Becasue we say so.")

    Hotel.main_menu()

from inputValidation import Validation


class Hotel:
    """Represents the hotel managing rooms, guests, and bookings."""
    def __init__(self, Room, Guest):
        self.Room = Room
        self.Guest = Guest

    def main_menu(self):
        print("\n\033[0;0mAvailable options:\n"
              "\033[95m1.\033[0;0m Rooms\n"
              "\033[95m2.\033[0;0m Bookings\n"
              "\033[95m3.\033[0;0m Guest Services\n"
              "\033[95m4.\033[0;0m Accounting\n"
              "\033[95m5.\033[0;0m Loyalty Programs\n"
              "\033[95m6.\033[0;0m Feedbacks\033[0;0m\n")
        selected_option = input("Select a number: ")
        status, value = Validation.validate_text_input(input_text=selected_option, min_value=1, max_value=6)
        if status == False:
            print(value)
            self.main_menu()
        else:
            print(chr(27) + "[2J")
            match value:
                case 1:
                    # Rooms
                    self.Room.rooms_menu(Hotel=self)
                case 2:
                    # Rooms
                    self.Guest.guest_menu(Hotel=self)
                case 3:
                    # Rooms
                    print("rooms")
                case 4:
                    # Rooms
                    print("rooms")
                case 5:
                    # Rooms
                    print("rooms")
                case 6:
                    # Rooms
                    print("rooms")



from inputValidation import Validation
from tabulate import tabulate  # For clean table formatting



class Guest:
    def __init__(self, conn):
        self.cursor = conn.cursor()
        self.conn = conn

    def guest_menu(self, Hotel):
        print("\n\033[0;0mAvailable options:\n"
              "\033[95m1.\033[0;0m List all Guests\n"
              "\033[95m2.\033[0;0m Find Guest\n"
              "\033[95m3.\033[0;0m Update Guest Information\n"
              "\033[95m4.\033[0;0m Main Menu\n")
        selected_option = input("Select a number: ")
        status, value = Validation.validate_text_input(input_text=selected_option, min_value=1, max_value=6)
        if status == False:
            print(value)
            self.guest_menu(Hotel)
        else:
            match value:
                case 1:
                    # Rooms
                    self.display_all_guests()
                    self.guest_menu(Hotel)

                case 2:
                    # Rooms
            self.lookup_guest_by_room()
            self.guest_menu(Hotel)
        case 3:
            self.update_guest_by_room()
            self.guest_menu(Hotel)

        case 4:
            # Rooms
            Hotel.main_menu()

# Function to display all guests and their room number in a table using tabulate
def display_all_guests(self):
    self.cursor.execute('''
        SELECT B.ROOM_ID, G.NAME, G.EMAIL, G.PHONE, G.LOYALTY_STATUS
        FROM GUESTS G
        JOIN BOOKINGS B ON G.GUEST_ID = B.GUEST_ID
    ''')
    guests = self.cursor.fetchall()
    headers = ["Room Number", "Name", "Email", "Phone", "Loyalty Status"]
    print(tabulate(guests, headers=headers, tablefmt="grid"))

# Function to insert guest data
def insert_guest(self, name: str, email: str, phone: str, loyalty_status: str):
    self.cursor.execute('INSERT INTO GUESTS (NAME, EMAIL, PHONE, LOYALTY_STATUS) VALUES (?, ?, ?, ?)',
                (name, email, phone, loyalty_status))
    self.conn.commit()

# Function to look up a guest by room number using user input and display with tabulate
def lookup_guest_by_room(self):
    room_number = input("Enter room number to look up guest: ")
    try:
        room_id = int(room_number)
        self.cursor.execute('''
            SELECT B.ROOM_ID, G.NAME, G.EMAIL, G.PHONE, G.LOYALTY_STATUS
            FROM GUESTS G
            JOIN BOOKINGS B ON G.GUEST_ID = B.GUEST_ID
            WHERE B.ROOM_ID = ?
        ''', (room_id,))
        guest = self.cursor.fetchone()
        if guest:
            headers = ["Room Number", "Name", "Email", "Phone", "Loyalty Status"]
            print(tabulate([guest], headers=headers, tablefmt="grid"))
        else:
            print("No guest found for that room.")
    except ValueError:
        print("Invalid room number entered.")

# Function to allow user to update a guest based on room number
def update_guest_by_room(self):
    room_number = input("Enter room number to update guest details: ")
    try:
        room_id = int(room_number)
        self.cursor.execute('''
            SELECT G.GUEST_ID, G.NAME, G.EMAIL, G.PHONE, G.LOYALTY_STATUS
            FROM GUESTS G
            JOIN BOOKINGS B ON G.GUEST_ID = B.GUEST_ID
            WHERE B.ROOM_ID = ?
        ''', (room_id,))
        result = self.cursor.fetchone()
        if result:
            guest_id, name, email, phone, loyalty_status = result
            print("Leave input blank and press Enter to keep current value.")
            new_name = input(f"Name [\033[95m{name}\033[0;0m]: ") or name
            new_email = input(f"Email [\033[95m{email}\033[0;0m]: ") or email
            new_phone = input(f"Phone [\033[95m{phone}\033[0;0m]: ") or phone
            new_loyalty = input(f"Loyalty Status [\033[95m{loyalty_status}\033[0;0m]: ") or loyalty_status

            self.cursor.execute('''
                UPDATE GUESTS
                SET NAME = ?, EMAIL = ?, PHONE = ?, LOYALTY_STATUS = ?
                WHERE GUEST_ID = ?
            ''', (new_name, new_email, new_phone, new_loyalty, guest_id))
            self.conn.commit()
            print("Guest information updated successfully.")
        else:
            print("No guest found for that room.")
    except ValueError:
        print("Invalid room number entered.")


    # Function to query guest reservation history placeholder (to be implemented with reservations table)
    def get_guest_reservation_history(guest_id: int):
        # Placeholder: This function would normally fetch reservation history linked to the guest
        return f"Reservation history for guest ID {guest_id} not yet implemented."

from typing import List, Tuple
from inputValidation import Validation
from tabulate import tabulate  # For clean table formatting


class Room:
    """Represents a hotel room with number, type, price, and bookings."""

    def __init__(self, conn):
        self.cursor = conn.cursor()

    def rooms_menu(self, Hotel):
        print("\n\033[0;0mRooms Menu Options:\n"
              "\033[95m1.\033[0;0m Available Rooms\n"
              "\033[95m2.\033[0;0m Booked Rooms\n"
              "\033[95m3.\033[0;0m Maintanance Rooms\n"
              "\033[95m4.\033[0;0m Search Room\n"
              "\033[95m5.\033[0;0m Main Menu\n")
        selected_option = input("Select a number: ")
        status, value = Validation.validate_text_input(input_text=selected_option, min_value=1, max_value=5)
        if status == False:
            print(value)
            self.rooms_menu(Hotel)
        else:
            print(chr(27) + "[2J")
            match value:
                case 1:
                    # Rooms
                    self.print_available_rooms()
                    self.rooms_menu(Hotel)
                case 2:
                    # Rooms
                    self.print_booked_rooms()
                    self.rooms_menu(Hotel)
                case 3:
                    # Rooms
                    self.print_maintanance_rooms()
                    self.rooms_menu(Hotel)
                case 4:
                    # Rooms
                    room_id = int(input("Please select a room number: "))
                    self.print_room(room_id)
                    self.rooms_menu(Hotel)

                case 5:
                    Hotel.main_menu()

                case 1:
                    # Rooms
                    print("rooms")


    def print_available_rooms(self):
        print("Available Rooms:")
        rooms = self.query_rooms_by_status('Available')
        headers = ['Room ID', 'Type', 'Amenities', 'Price', 'Status']
        print(tabulate(rooms, headers=headers, tablefmt='grid'))


    def print_booked_rooms(self):
        print("Booked Rooms:")
        rooms = self.query_rooms_by_status('Occupied')
        headers = ['Room ID', 'Type', 'Amenities', 'Price', 'Status']
        print(tabulate(rooms, headers=headers, tablefmt='grid'))

    def print_maintanance_rooms(self):
        print("Maintenance Rooms:")
        rooms = self.query_rooms_by_status('Maintenance')
        headers = ['Room ID', 'Type', 'Amenities', 'Price', 'Status']
        print(tabulate(rooms, headers=headers, tablefmt='grid'))

    def print_room(self, room_id):
        print(f"Rooms {room_id}:")
        self.cursor.execute('SELECT * FROM ROOMS WHERE ID = ?', (room_id,))
        room = self.cursor.fetchone()
        if room:
                headers = ['Room ID', 'Type', 'Amenities', 'Price', 'Status']
                print(tabulate([room], headers=headers, tablefmt='grid'))
        else:
                print("Room not found.")


    def query_rooms_by_status(self, status: str) -> List[Tuple[int, str, str, float, str]]:
        self.cursor.execute('SELECT * FROM ROOMS WHERE STATUS = ?', (status,))
        return self.cursor.fetchall()


class Feedback:
    def __init__(self, conn):
        self.cursor = conn.cursor()
        self.conn = conn

    # Function to submit feedback after stay
    def submit_feedback(self, guest_id: int, room_id: int, rating: int, comments: str):
        self.cursor.execute('''
                INSERT INTO FEEDBACK (GUEST_ID, ROOM_ID, RATING, COMMENTS)
                VALUES (?, ?, ?, ?)
        ''', (guest_id, room_id, rating, comments))
        self.conn.commit()
```

```
class ServiceRequest:
    def __init__(self, conn):
        self.cursor = conn.cursor()
        self.conn = conn

    # Function to submit a guest service request
    def submit_request(self, guest_id: int, room_id: int, request_type: str, description: str):
        self.cursor.execute('''
            INSERT INTO REQUESTS (GUEST_ID, ROOM_ID, REQUEST_TYPE, REQUEST_DESCRIPTION)
            VALUES (?, ?, ?, ?)
        ''', (guest_id, room_id, request_type, description))
        self.conn.commit()

class Booking:

    def __init__(self, conn):
        self.cursor = conn.cursor()

from datetime import datetime


class Payment:

    def __init__(self, conn):
        self.cursor = conn.cursor()
        self.conn = conn

    # Function to create invoice upon guest checkout
    def create_invoice(self, booking_id: int, additional_charges: float = 0.0, discounts: float = 0.0,
                       payment_method: str = 'Credit Card'):
        self.cursor.execute('''
            SELECT B.ROOM_ID, B.CHECK_IN_DATE, B.CHECK_OUT_DATE, R.PRICE
            FROM BOOKINGS B
            JOIN ROOMS R ON B.ROOM_ID = R.ID
            WHERE B.BOOKING_ID = ?
        ''', (booking_id,))
        result = self.cursor.fetchone()
        if result:
            room_id, check_in, check_out, nightly_rate = result
            d1 = datetime.strptime(check_in, '%Y-%m-%d')
            d2 = datetime.strptime(check_out, '%Y-%m-%d')
            num_nights = (d2 - d1).days
            total_amount = max((nightly_rate * num_nights + additional_charges - discounts), 0)

            self.cursor.execute('''
                INSERT INTO ACCOUNTING (BOOKING_ID, NIGHTLY_RATE, NUM_NIGHTS, ADDITIONAL_CHARGES,
DISCOUNTS, TOTAL_AMOUNT, PAYMENT_METHOD)
                VALUES (?, ?, ?, ?, ?, ?, ?)
            ''', (booking_id, nightly_rate, num_nights, additional_charges, discounts, total_amount, payment_method))

            self.cursor.execute('UPDATE ROOMS SET STATUS = ? WHERE ID = ?', ('Available', room_id))
            self.conn.commit()
```

Test Cases

1. **Main Menu Display Test**
   Description:
   This test confirms that the program starts correctly by displaying the ASCII art header and the main menu options. It ensures that the user sees all available options (Rooms, Bookings, Guest Services, Accounting, Loyalty Programs, Feedback).
   Steps:
   - Run the program.
   - Observe the printed ASCII art and the main menu options.
   Expected Output:
   - The ASCII art header is printed.
   - The main menu displays options:
     - 1. Rooms
     - 2. Bookings
     - 3. Guest Services
     - 4. Accounting
     - 5. Loyalty Programs
     - 6. Feedback
2. **Rooms Menu Navigation Test**
   Description:
   This test verifies that selecting option “1. Rooms” from the main menu correctly navigates the user to the Rooms Menu and displays the available room-related options.
   Steps:
   - From the main menu, input “1” to select Rooms.
   - Verify that the Rooms Menu is displayed with its options (Available Rooms, Booked Rooms, Maintenance Rooms, Search Room, and Main Menu).
   Expected Output:
   - A clear list of Rooms Menu options is printed.

3. **Guest Menu Navigation and Display Test**
   Description:
   This test ensures that choosing the guest management option (via option "2" from the main menu) navigates to the Guest Menu, and that the "List all Guests" function correctly displays a table of guest information.
   Steps:
   - From the main menu, input "2" to enter Guest Services.
   - In the Guest Menu, select option "1" to list all guests. **Expected Output:**
   - A table with columns such as "Room Number", "Name", "Email", "Phone", and "Loyalty Status" is printed.

4. **Room Lookup Test**
   Description:
   This test verifies that using the "Search Room" option (option "4" in the Rooms Menu) with a valid room number returns the correct room details in a formatted table.
   Steps:
   - From the Rooms Menu, select option "4."
   - Enter a valid room number (e.g., 101). **Expected Output:**
   - A table showing the details for Room 101 if it exists, or a "Room not found" message otherwise.

5. **Update Guest Information Test**
   Description:
   This test confirms that the guest information update functionality works correctly. The user is prompted to update details and can keep the current value by leaving the input blank.
   Steps:
   - In the Guest Menu, select option "3" for updating guest information.
   - Enter a valid room number, then change one or more fields (e.g., name, email) and leave others unchanged. **Expected Output:**
   - A message stating "Guest information updated successfully."
   - When listing all guests again, the updated details are reflected.

6. **Payment Invoice Generation Test**
   Description:
   This test simulates payment processing by calling the create_invoice method from the Payment class. It verifies that the invoice is correctly calculated and inserted into the accounting table, and that the room status is updated to "Available."
   Steps:

- Simulate a payment by invoking the create_invoice method with a valid booking ID, additional charges, and discounts.
- Verify the output (via a printed confirmation or database check) and ensure that the room's status is updated. **Expected Output:**
- An invoice is created with the correct total amount.
- The room's status is changed to "Available."

7. **Service Request Submission Test**
   Description:
   This test confirms that submitting a guest service request using the submit_request method properly records the request in the database.
   Steps:
   - Invoke the submit_request method with test values (guest ID, room ID, request type such as "Room Service", and a description). **Expected Output:**
   - The service request is recorded (this may be verified via a database query or log message).

8. **Feedback Submission Test**
   Description:
   This test verifies that the feedback functionality works by ensuring that a feedback record is successfully created when the submit_feedback method is called.
   Steps:
   - Call the submit_feedback method with test parameters (guest ID, room ID, rating, and comments). **Expected Output:**
   - The feedback is recorded in the system.

Output:

```
    ___   ___  ___   _____ ______    _______    ___   ________   ________   ___  ___
   |\  \ |\  \|\  \ |\   _ \  _   \ |\  ___ \  |\  \ |\   __  \ |\   __  \ |\  \|\  \
   \ \  \\ \  \\\  \\ \  \\\__\ \  \\ \   __/| \ \  \\ \  \|\  \\ \  \|\  \\ \  \\\  \
 __ \ \  \\ \  \\\  \\ \  \\|__| \  \\ \  \_|/__\ \  \\ \   _  _\\ \   __  \\ \   __  \
|\  \\_\  \\ \  \\\  \\ \  \    \ \  \\ \  \_|\ \\ \  \\ \  \\  \|\ \  \ \  \\ \  \ \  \
\ \________\\ \_______\\ \__\    \ \__\\ \_______\\ \__\\ \__\\ _\ \ \__\ \__\\ \__\ \__\
 \|________| \|_______| \|__|     \|__| \|_______| \|__| \|__|\|__| \|__|\|__| \|__|\|__|

 ___  ___   ________   _________   _______    ___
|\  \|\  \ |\   __  \ |\___   ___\|\  ___ \  |\  \
\ \  \\\  \\ \  \|\  \\|___ \  \_|\ \   __/| \ \  \
 \ \   __  \\ \  \\\  \    \ \  \ \ \  \_|/__\ \  \
  \ \  \ \  \\ \  \\\  \    \ \  \ \ \  \_|\ \\ \  \____
   \ \__\ \__\\ \_______\    \ \__\ \ \_______\\ \_______\
    \|__|\|__| \|_______|     \|__|   \|_______| \|_______|

Jumeirah Hotel by Abood

Best Hotel Ever, Becasue we say so.


Available options:

1. Rooms

2. Bookings

3. Guest Services
```

```
4. Accounting

5. Loyalty Programs

6. Feedbacks


Select a number: 2



Available options:

1. List all Guests

2. Find Guest

3. Update Guest Information

4. Main Menu


Select a number: 2

Enter room number to look up guest: 101

+---------------+-------------------+------------------------------+------------+------------------+

|   Room Number | Name              | Email                        |      Phone | Loyalty Status   |

+===============+===================+==============================+
============+==================+

|           101 | Latifa Al Rashedi | latifa.alrashedi998@gmail.com | 0508419137 | Bronze           |

+---------------+-------------------+------------------------------+------------+------------------+


Available options:

1. List all Guests
```

2. Find Guest

3. Update Guest Information

4. Main Menu

Select a number:

GitGub Link:

https://github.com/Aboood2220/Royal-Stay-Hotel-Management-System

Summary:

In this project I learned to use Python classes and objects to build a hotel system. I found out how a hotel links to rooms and guests. Drawing a UML diagram helped me plan before coding which made things easier.

Writing the code was hard sometimes especially making all the classes work together. But I learned why keeping code neat.

This project showed me how to use object-oriented programming to make a system step by step and now I feel better about using these skills later whenever needed.