**1. Menu Management**

This feature will allow the restaurant admin to add, edit, and delete menu items.

**Implementation:**

1. **Menu Manager Class:** Create a MenuManager class in the Business Logic Layer that will encapsulate all the menu management operations.
2. **Methods:** Define methods like addMenuItem(MenuItem item), editMenuItem(MenuItem item), and deleteMenuItem(int itemId) to perform the desired actions.
3. **Data Access:** Use a DAO (Data Access Object) class to interact with the database (SQLite) and store/retrieve menu item data.
4. **Admin Interface:** In the admin interface, provide input fields and buttons for adding, editing, and deleting menu items. Connect these UI elements to the MenuManager methods.

**Example Code:**

public class MenuManager {

private DAO dao;

public MenuManager(DAO dao) {

this.dao = dao;

}

public void addMenuItem(MenuItem item) {

dao.createMenuItem(item);

}

public void editMenuItem(MenuItem item) {

dao.updateMenuItem(item);

}

public void deleteMenuItem(int itemId) {

dao.deleteMenuItem(itemId);

}

}

**2. Order Placement**

This feature allows customers to place orders, selecting items from the menu and providing their delivery information.

**Implementation:**

1. **Order Manager Class:** Create an OrderManager class in the Business Logic Layer to handle order creation and related logic.
2. **Methods:** Define methods like createOrder(Order order) and placeOrder(Customer customer, List<MenuItem> items, Address address) to handle the order placement process.
3. **UI:** In the customer-facing UI, provide a menu browsing interface with item selection and quantity controls. Include input fields for delivery address and contact information. Connect these UI elements to the OrderManager methods.

**Example Code:**

public class OrderManager {

private DAO dao;

public OrderManager(DAO dao) {

this.dao = dao;

}

public void createOrder(Order order) {

dao.createOrder(order);

}

public void placeOrder(Customer customer, List<MenuItem> items, Address address) {

Order order = new Order(customer, items, address);

createOrder(order);

}

}

**3. Order Status Management**

The restaurant admin should be able to view and update the status of orders.

**Implementation:**

1. **Order Manager:** Enhance the OrderManager class with methods to retrieve orders, update their status, and manage the order lifecycle.
2. **Admin Interface:** In the admin interface, display a list of incoming orders with their current status. Provide buttons or controls to update the status (e.g., "preparing," "out for delivery," "completed").

**Example Code:**

public class OrderManager {

private DAO dao;

public OrderManager(DAO dao) {

this.dao = dao;

}

public List<Order> getOrders() {

return dao.getAllOrders();

}

public void updateOrderStatus(Order order, OrderStatus status) {

order.setStatus(status);

dao.updateOrder(order);

}

}

**4. Customer Notifications**

Customers should receive notifications about their order status, such as when the order is placed, confirmed, prepared, and out for delivery.

**Implementation:**

1. **Notification Service:** Create a notification service that can send push notifications to users.
2. **Order Manager:** In the OrderManager, integrate the notification service to send notifications at appropriate stages of the order lifecycle.