Usage of Cohesion and Coupling in the Code :

Cohesion :

Cohesion refers to the degree to which elements within a module work together to fulfill a single, well-defined purpose. High cohesion means that elements are closely related and focused on a single purpose, while low cohesion means that elements are loosely related and serve multiple purposes.

Usage of Cohesion :

1. Logical Cohesion :

   Though both the functions are with respecting to updating , updating here are two entities(quantity and price)

```
public void updateMenuItemPrice(String name, int price) {
    for (MenuItem item : menuCard) {
        if (item.getName().equalsIgnoreCase(name)) {
            item.setPrice(price);
            System.out.println(name + "'s price updated successfully in menu!");
            return;
        }
    }
    System.out.println("ERROR : Item not found in menu!");
}

public void updateMenuItemQuantity(String name, int quantity) {
    for (MenuItem item : menuCard) {
        if (item.getName().equalsIgnoreCase(name)) {
            item.setQuantity(quantity);
            System.out.println(name + "'s quantity updated successfully in menu!");
            return;
        }
    }
    System.out.println("ERROR : Item not found in menu!");
}
```

2. Procedural Cohesion :

   Here the Menucard is checked for existence ,only if it is present the next subsequent steps are proceeded with.

```
public void addMenuItem(String name, double price, int quantity) {
    if (menuCard == null)
        menuCard = new MenuCard();
    menuCard.addMenuItem(new MenuItem(name, price, quantity, this.username));
}
```

3. Sequential Cohesion :

Here the (initial - quantity) serves as the input to the menu-card of for it to be updated .

There is a good difference between the procedural and sequential cohesion ,

- In Procedural cohesion, the sequential flow refers to the specific order in which functions or operations must be executed.
- Sequential cohesion involves activities that are interconnected, where the output of one activity serves as the input for the next activity.

```
public int updateItem(String name, String adminName, int menuQuantity) {
    for (Cart cart : carts) {
        if (cart.getAdminName().equalsIgnoreCase(adminName)) {
            for (UserItem userItem : cart.getItems()) {
                if (userItem.getName().equals(name)) {
                    int initial = userItem.getQuantity();
                    System.out.println("Enter the quantity to be updated");
                    int quantity = Integer.parseInt(System.console().readLine());
                    if (initial - quantity + menuQuantity < 0) {
                        System.out.println("ERROR : Quantity not available!");
                        return 0;
                    }
```

```
                    userItem.setQuantity(quantity);
                    System.out.println(name + "'s quantity updated successfully in cart!");
                    return initial - quantity;
                }
            }
        }
```

4 . <u>Communicational Cohesion :</u>

Since both the member functions acts only on the menu card (same data structure)

```
class MenuCard {
    private List<MenuItem> menuCard;

    MenuCard() {
        menuCard = new ArrayList<>();
    }

    public void addMenuItem(MenuItem item) {
        menuCard.add(item);
        System.out.println(item.getName() + " added to menu successfully!");
    }

    public void removeMenuItem(String name) {
        for (MenuItem menuItem : menuCard) {
            if (menuItem.getName().equals(name)) {
                menuCard.remove(menuItem);
                System.out.println(name + " removed from menu successfully!");
                return;
            }
        }
        System.out.println("ERROR : Item not found in menu!");
    }
```

Coupling :

Coupling refers to the degree of interdependence between software modules. High coupling means that modules are closely connected and changes in one module may affect other modules. Low coupling means that modules are independent, and changes in one module have little impact on other modules.

Usage of Coupling :

Data Coupling :

Since the number of parameters exceeds the limit of 3 data coupling is incorporated here.

There is always a trade-off between Data Coupling and Stamp Coupling, where using one will eliminate the other.

```
class MenuItem {
   private String name;
   private double price;
   private int quantity;
   private String adminName;

   MenuItem(String name, double price, int quantity, String adminName) {
      this.name = name;
```

Stamp Coupling :

Since here the Menu Item Module is passed as a parameter to Menu Card class instead of its attributes Stamp Coupling is used here.

```
public void addMenuItem(MenuItem item) {
   menuCard.add(item);
   System.out.println(item.getName() + " added to menu successfully!");
}
```

Control Coupling :

Since both the updateMenuItemPrice function and updateMenuItemQuantity function of the MenuCard Class has access to the common Menu-Card data structure,it follows Control Coupling.

Control Coupl

```
public void updateMenuItemPrice(String name, int price) {
   for (MenuItem item : menuCard) {
      if (item.getName().equalsIgnoreCase(name)) {
         item.setPrice(price);
         System.out.println(name + "'s price updated successfully in menu!");
         return;
      }
   }
   System.out.println("ERROR : Item not found in menu!");
}

public void updateMenuItemQuantity(String name, int quantity) {
   for (MenuItem item : menuCard) {
      if (item.getName().equalsIgnoreCase(name)) {
         item.setQuantity(quantity);
         System.out.println(name + "'s quantity updated successfully in menu!");
         return;
      }
   }
   System.out.println("ERROR : Item not found in menu!");
}
```

Here the control to different modules is passed with the help of switch-case statements.

-

```
while (adminRun.equalsIgnoreCase("Y")) {

   System.out.println();
   System.out.println("-------------------------------------------------------");
   System.out.println("Admin Choices: ");
   System.out.println("-------------------------------------------------------");
   System.out.println("1. To view menu");
   System.out.println("2. To add a menu item");
   System.out.println("3. To delete a menu item");
   System.out.println("4. To update price");
   System.out.println("5. To update quantity");
   System.out.println("6. To view all users");
   System.out.println("7. To view all orders");
   System.out.println("8. To view all feedbacks");
   System.out.println("9. To remove the order");
   System.out.println("-------------------------------------------------------");
   System.out.println();

   System.out.println("Enter your choice: ");
   int adminChoice = Integer.parseInt(System.console().readLine());

   switch (adminChoice) {
      case 1:
         admin.viewMenuCard();
         break;
      case 2:
         System.out.println();
         System.out.println("Enter the name of the menu item");
         String nameAdd = System.console().readLine();
         System.out.println("Enter the price of the menu item");
         double price = Double.parseDouble(System.console().readLine());
         System.out.println("Enter the quantity of the menu item");
         int quantity = Integer.parseInt(System.console().readLine());
         admin.addMenuItem(nameAdd, price, quantity);
         break;
```