Scenario:

This Java-based Food Ordering System allows users to browse and order food online. Customers can choose between Takeout (pickup, no delivery fees) or Delivery (driver assigned, extra fee) and put into discount codes. Menu items are loaded from a file using File I/O. The system includes an Admin portal for managing menu items, setting discount codes, and processing orders. Drivers can log in to accept delivery orders, mark them as delivered, or cancel accepted orders.

Expected Outputs:

- Customers can log into the system.
- They can browse menu items loaded from a text file.
- Customers can place either a Takeout or Delivery order.
- They can apply valid discount codes during checkout.
- After ordering, customers can view their order history.

- Admins access a dedicated admin portal.
- The viewDashboard() method shows a custom interface.
- Admins can add, update, or delete menu items.
- They can manage discount codes.
- Admins process customer orders using the processOrder() method.
- They can save menu updates and logs to text files.

- Drivers log in to see available delivery orders.
- They can accept orders, mark them as delivered, or cancel them.
- The processOrder() method lets drivers update order status.
- Drivers can also view their delivery history.

- The system uses an abstract Order class.
- There are two subclasses: TakeoutOrder and DeliveryOrder.
- Orders include an ID, customer info, items, and total price.
- Delivery orders include a delivery fee and are assigned to a driver.
- Takeout orders are scheduled for pickup without delivery steps.

- Menu items are managed by the MenuItem class.
- This class implements Comparable to sort by name and genre.
- The PriceComparator class allows sorting by price.
- Menu items are stored in a List and filtered using streams.

- The OrderManager class handles file input/output.
- It loads menu data at startup and saves order histories.
- All admin changes and orders are saved using this class.

- Testing is done in TestClass using JUnit.

Design Paradigm:

The project will demonstrate:

Class hierarchies:
(e.g. User → Customer, Admin, Driver)
(e.g. Order → TakeoutOrder, DeliveryOrder)

Polymorphism:
Runtime polymorphism:
- Customer.placeOrder() handles both TakeoutOrder and DeliveryOrder
- User.viewDashboard() is overridden in each subclass

Interfaces
OrderProcessor interface with a processOrder method which will be implemented by both Admin and Driver classes to manage or fulfill orders

Collections:
- Used to maintain ordered collections of objects, such as:
    - Lists to store the menu items loaded from a file.
    - Lists to store all past and current orders (can be saved/loaded from a file).
    - Lists to store active discount codes.

Text I/O:
- Used in the OrderManager class:
    - Load menu items from a text file
    - Save order history and logs to files

Stream:
- Stream processing for filtering/searching menu items

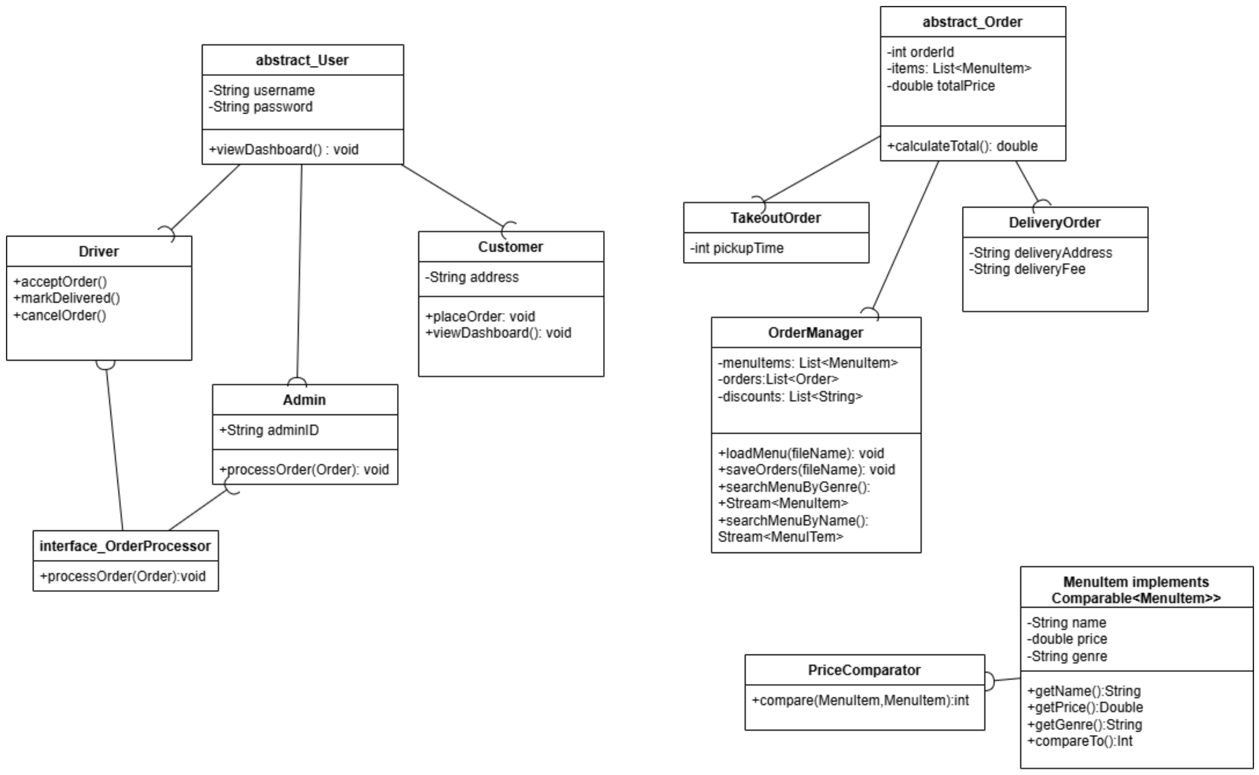
Tests:
Tests will be done in TestClass using JUnit

Comparable and Comparator:
- MenuItem implements Comparable<MenuItem> to sort by name
- PriceComparator implements Comparator<MenuItem> to sort menu items by price

Deliverable 2 Implementation:

- Class structure for User, Customer, Admin, Driver
- Abstract Order class and concrete TakeoutOrder, DeliveryOrder
- Interface OrderProcessor
- MenuItem class with Comparable implementation
- OrderManager class with file reading to load menu
- Basic Admin dashboard functionality

UML CLASS DIAGRAM:


abstract_User
-String username
-String password
+viewDashboard() : void
Driver
+acceptOrder()
+markDelivered()
+cancelOrder()
Customer
-String address
+placeOrder: void
+viewDashboard(): void
Admin
+String adminID
+processOrder(Order): void
interface_OrderProcessor
+processOrder(Order):void
abstract_Order
-int orderId
-items: List<MenuItem>
-double totalPrice
+calculateTotal(): double
TakeoutOrder
-int pickupTime
DeliveryOrder
-String deliveryAddress
-String deliveryFee
OrderManager
-menuItems: List<MenuItem>
-orders:List<Order>
-discounts: List<String>
+loadMenu(fileName): void
+saveOrders(fileName): void
+searchMenuByGenre():
+Stream<MenuItem>
+searchMenuByName():
Stream<MenulTem>
MenuItem implements Comparable<MenuItem>>
-String name
-double price
-String genre
+getName():String
+getPrice():Double
+getGenre():String
+compareTo():Int
PriceComparator
+compare(MenuItem,MenuItem):int