**Restaurant Management System** that includes **Interfaces**, **Implementations**, **Collections**, **Exceptions**, **Streams**, and **Serialization**.

In this system, I’ll use:

* **Interfaces** to define the basic behavior of the restaurant (e.g., adding menu items, placing orders).
* **Implementations** to implement these behaviors concretely (e.g., managing menu items, orders).
* **Collections** to store items (like menu items and orders).
* **Exceptions** to handle edge cases (e.g., invalid menu item or item not found).
* **Streams** for filtering and processing orders.
* **Serialization** to persist data (e.g., save/load menu and orders).

**1. Define the MenuItem Class (Serializable)**

import java.io.Serializable;

public class MenuItem implements Serializable {

private String name;

private double price;

private String description;

public MenuItem(String name, double price, String description) {

this.name = name;

this.price = price;

this.description = description;

}

public String getName() {

return name;

}

public double getPrice() {

return price;

}

public String getDescription() {

return description;

}

@Override

public String toString() {

return String.format("Name: %s, Price: %.2f, Description: %s", name, price, description);

}

}

**2. Define the Order Class (Serializable)**

import java.io.Serializable;

import java.util.List;

public class Order implements Serializable {

private List<MenuItem> items;

private double totalPrice;

public Order(List<MenuItem> items) {

this.items = items;

this.totalPrice = items.stream().mapToDouble(MenuItem::getPrice).sum();

}

public List<MenuItem> getItems() {

return items;

}

public double getTotalPrice() {

return totalPrice;

}

@Override

public String toString() {

StringBuilder orderDetails = new StringBuilder("Order Details:\n");

items.forEach(item -> orderDetails.append(item).append("\n"));

orderDetails.append("Total Price: ").append(totalPrice);

return orderDetails.toString();

}

}

**3. Define the RestaurantOperations Interface**

This interface will define the contract for operations like adding/removing menu items, creating orders, etc.

import java.io.IOException;

import java.util.List;

public interface RestaurantOperations {

void addMenuItem(MenuItem item);

void removeMenuItem(String name);

List<MenuItem> getMenu();

Order createOrder(List<String> itemNames) throws ItemNotFoundException;

void saveData(String filename) throws IOException;

void loadData(String filename) throws IOException, ClassNotFoundException;

}

**4. Define the Restaurant Class (Implementation of RestaurantOperations)**

This class will implement the RestaurantOperations interface. It will use collections to store menu items and orders and handle the core operations.

import java.io.\*;

import java.util.\*;

import java.util.stream.Collectors;

public class Restaurant implements RestaurantOperations {

private List<MenuItem> menu;

private List<Order> orders;

public Restaurant() {

menu = new ArrayList<>();

orders = new ArrayList<>();

}

@Override

public void addMenuItem(MenuItem item) {

menu.add(item);

}

@Override

public void removeMenuItem(String name) {

menu.removeIf(item -> item.getName().equalsIgnoreCase(name));

}

@Override

public List<MenuItem> getMenu() {

return menu;

}

@Override

public Order createOrder(List<String> itemNames) throws ItemNotFoundException {

List<MenuItem> orderedItems = menu.stream()

.filter(item -> itemNames.contains(item.getName()))

.collect(Collectors.toList());

if (orderedItems.size() != itemNames.size()) {

throw new ItemNotFoundException("Some items were not found on the menu.");

}

Order order = new Order(orderedItems);

orders.add(order);

return order;

}

@Override

public void saveData(String filename) throws IOException {

try (ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream(filename))) {

out.writeObject(menu);

out.writeObject(orders);

}

}

@Override

public void loadData(String filename) throws IOException, ClassNotFoundException {

try (ObjectInputStream in = new ObjectInputStream(new FileInputStream(filename))) {

menu = (List<MenuItem>) in.readObject();

orders = (List<Order>) in.readObject();

}

}

// View orders (optional helper method)

public void viewOrders() {

if (orders.isEmpty()) {

System.out.println("No orders placed yet.");

} else {

orders.forEach(System.out::println);

}

}

// View the menu (optional helper method)

public void displayMenu() {

if (menu.isEmpty()) {

System.out.println("No items in the menu.");

} else {

menu.forEach(System.out::println);

}

}

}

**5. Define the ItemNotFoundException Class**

This custom exception will be thrown when an item is not found in the menu during order creation.

public class ItemNotFoundException extends Exception {

public ItemNotFoundException(String message) {

super(message);

}

}

**6. Main Class to Test the Restaurant Management System**

This class will simulate operations like adding items to the menu, placing orders, and saving/loading data.

import java.io.IOException;

import java.util.Arrays;

import java.util.List;

import java.util.Scanner;

public class RestaurantManagementSystem {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

Restaurant restaurant = new Restaurant();

// Add menu items

restaurant.addMenuItem(new MenuItem("Pizza", 12.99, "Delicious cheese pizza"));

restaurant.addMenuItem(new MenuItem("Pasta", 8.99, "Pasta with marinara sauce"));

restaurant.addMenuItem(new MenuItem("Burger", 5.99, "Beef burger with fries"));

// Display menu

System.out.println("Menu:");

restaurant.displayMenu();

// Create an order

System.out.println("\nCreating an order...");

try {

List<String> itemNames = Arrays.asList("Pizza", "Burger");

Order order = restaurant.createOrder(itemNames);

System.out.println(order);

} catch (ItemNotFoundException e) {

System.out.println("Error: " + e.getMessage());

}

// View all orders

System.out.println("\nAll Orders:");

restaurant.viewOrders();

// Save menu and orders to file

try {

restaurant.saveData("restaurant\_data.ser");

System.out.println("\nData saved to file.");

} catch (IOException e) {

System.out.println("Error saving data: " + e.getMessage());

}

// Load menu and orders from file

try {

restaurant.loadData("restaurant\_data.ser");

System.out.println("\nData loaded from file.");

restaurant.displayMenu();

restaurant.viewOrders();

} catch (IOException | ClassNotFoundException e) {

System.out.println("Error loading data: " + e.getMessage());

}

}

}

**Explanation of Features**

1. **Interface (RestaurantOperations)**:
   * This defines the basic operations that any class (such as Restaurant) must implement, such as adding/removing menu items, creating orders, saving/loading data, etc.
2. **Implementation (Restaurant)**:
   * The Restaurant class implements the RestaurantOperations interface and provides concrete logic for adding/removing menu items, creating orders, saving/loading data, etc.
3. **MenuItem Class (Serializable)**:
   * Each MenuItem is an object that contains the name, price, and description of the item. This class is **Serializable** so we can save/load it.
4. **Order Class (Serializable)**:
   * Represents an order made by the customer. It holds the list of MenuItems and calculates the total price.
5. **ItemNotFoundException**:
   * A custom exception thrown when an item in the order is not found in the menu.
6. **Serialization**:
   * We use ObjectOutputStream and ObjectInputStream to save/load both the menu and orders.
7. **Streams**:
   * **Streams** are used to filter the menu based on the list of ordered items and calculate the total price.

**Sample Output**

Menu:

Name: Pizza, Price: 12.99, Description: Delicious cheese pizza

Name: Pasta, Price: 8.99, Description: Pasta with marinara sauce

Name: Burger, Price: 5.99, Description: Beef burger with fries

Creating an order...

Order Details:

Name: Pizza, Price: 12.99, Description: Delicious cheese pizza

Name: Burger, Price: 5.99, Description: Beef burger with fries

Total Price: 18.98

All Orders:

Order Details:

Name: Pizza, Price: 12.99, Description: Delicious cheese pizza

Name: Burger, Price: 5.99, Description: Beef burger with fries

Total Price: 18.98

Data saved to file.

Data loaded from file.

Name: Pizza, Price: 12.99, Description: Delicious cheese pizza

Name: Pasta, Price: 8.99, Description: Pasta with marinara sauce

Name: Burger, Price: 5.99, Description: Beef burger with fries

Order Details:

Name: Pizza, Price: 12.99, Description: Delicious cheese pizza

Name: Burger, Price: 5.99, Description: Beef burger with fries

Total Price: 18.98