**Case Study Document**

**Java Food Delivery System**

**Objective:**

You will develop a **menu-driven Food Delivery System** console application in Java that simulates a real-world food ordering and delivery process. The goal is to apply a wide range of Java programming concepts in a practical scenario.

**Problem Statement:**

You are tasked with building a simple but robust Food Delivery System that manages different types of food items, allows users to view, add, update, and delete menu items, and simulate the food delivery process. The system should store menu data persistently, allow sorting and searching, and handle exceptional cases gracefully.

**Learning Goals:**

By completing this case study, you will:

* Design and implement **abstract classes** and **inheritance** to represent different food categories.
* Use **arrays and collections** (ArrayList) to manage menu items.
* Perform **CRUD operations** (Create, Read, Update, Delete) on food items.
* Implement **file I/O** to save and load menu data from a text file.
* Create and throw **custom exceptions** for error handling.
* Apply **threading** to simulate asynchronous food delivery.
* Use **static variables and methods** to maintain common properties.
* Explore **lambda expressions** and **Stream API** to filter, sort, and manipulate collections efficiently.

**Functional Requirements:**

Your Food Delivery System should support the following menu-driven options:

1. **View all food items**  
   Display a list of all available food items.
2. **Add a new food item**  
   Add food items to the menu with appropriate details.
3. **Update existing food item**  
   Modify details of an existing food item by its ID.
4. **Delete a food item**  
   Remove a food item from the menu by its ID.
5. **Search food items**  
   Search and filter food items by name or category.
6. **Sort food items**  
   Sort food items by price or name using lambda and streams.
7. **Save menu to file**  
   Save the current menu into a text file for persistence.
8. **Load menu from file**  
   Load menu items from the file into the application.
9. **Simulate food delivery**  
   Use a separate thread to simulate the delivery process asynchronously.
10. **Exit the application**

**Non-Functional Requirements:**

* **Exception handling:** The application must handle invalid inputs, missing items, and file errors with meaningful messages.
* **Code organization:** Apply object-oriented principles with proper class structure and encapsulation.
* **User experience:** Provide clear console instructions and responses.
* **Performance:** Efficiently handle collections and file operations using streams and proper data structures.

**Class Design with Properties & Methods**

**1. Abstract Class: FoodItem**

Represents a general food item with common properties.

**Properties:**

* int id — Unique identifier for the food item
* String name — Name of the food item
* String category — Category like "Main Course", "Dessert", "Beverage"
* double price — Price of the food item
* int quantity — Quantity available or ordered

**Methods:**

* abstract void displayDetails() — Display food item details (to be implemented by subclasses)
* int getId(), void setId(int id)
* String getName(), void setName(String name)
* String getCategory(), void setCategory(String category)
* double getPrice(), void setPrice(double price)
* int getQuantity(), void setQuantity(int quantity)

**2. Concrete Classes: (Inheritance from FoodItem)**

**a) PermanentFoodItem**

Represents permanent food menu items that are always available.

**Additional Properties:**  
(None beyond FoodItem fields unless needed)

**Methods:**

* void displayDetails() — Override to show details specific to permanent items

**b) TemporaryFoodItem**

Represents temporary or seasonal food items.

**Additional Properties:**

* String availabilityPeriod — Dates or duration for which this item is available

**Methods:**

* void displayDetails() — Override to include availability period

**3. Class: FoodMenu**

Manages a collection of FoodItem objects.

**Properties:**

* ArrayList<FoodItem> foodList — List holding food items

**Methods:**

* void addFoodItem(FoodItem item)
* void updateFoodItem(int id, FoodItem updatedItem)
* void deleteFoodItem(int id)
* FoodItem searchFoodItemById(int id)
* List<FoodItem> searchFoodItemsByName(String name)
* List<FoodItem> searchFoodItemsByCategory(String category)
* List<FoodItem> getAllFoodItems()
* List<FoodItem> sortFoodItemsByPrice(boolean ascending)
* List<FoodItem> sortFoodItemsByName()

**4. Class: FileManager**

Handles file input/output operations.

**Properties:**

* String filePath — Path of the file to save/load data

**Methods:**

* void saveMenuToFile(List<FoodItem> foodItems)
* List<FoodItem> loadMenuFromFile()

**5. Class: DeliveryThread (extends Thread)**

Simulates asynchronous delivery process.

**Properties:**

* FoodItem item — Food item to be delivered
* int deliveryTime — Estimated delivery time in seconds

**Methods:**

* void run() — Override to simulate delivery delay and print status

**6. Class: CustomException (extends Exception)**

Handles application-specific exceptions like invalid inputs or file errors.

**Properties:**

* String message

**Methods:**

* Constructor with message parameter
* String getMessage()

**7. Utility Class: FoodUtils**

Contains static utility methods for common operations.

**Methods:**

* static void displayFoodList(List<FoodItem> items)
* static List<FoodItem> filterByCategory(List<FoodItem> items, String category)
* static List<FoodItem> filterByName(List<FoodItem> items, String name)
* static List<FoodItem> sortByPrice(List<FoodItem> items, boolean asc)
* static List<FoodItem> sortByName(List<FoodItem> items)

**Additional Notes:**

* You are free to design the classes and their properties as you see fit.
* Ensure that the system is extendable to add new food categories in future.
* Use appropriate Java features covered in your training to implement the system.